QUANTCONNECT DOCUMENTATION

Learn to use QuantConnect and Explore Our Features

WRITING ALGORITHMS

Learn tools you need to build algorithmic trading strategies

Build on a mature, flexible, feature-complete API managing billions of dollars capital, and used by 5,000+ investors every month.

Table of Content

Key Concepts

Key Concepts

Getting Started

Introduction

Quantitative trading is a method of trading where computer programs execute a set of defined trading rules in an automated fashion. Quants take a scientific approach to trading, applying concepts from mathematics, time series analysis, statistics, computer science, and machine learning. Compared to discretionary traders, quants can respond faster to new information and are at less risk to their emotions during trades. Since quants can concurrently trade many strategies while discretionary traders only have the mental bandwidth to trade a small number of concurrent strategies, quant traders can have more diversified portfolios.

Learn Programming

We aim to make it as easy as possible to use QuantConnect, but you still need to be able to program. The following table provides some resources to get you started:

Language	Type	Name	Producer
C#	Video	C# Fundamentals for Absolute Beginners	Microsoft
C#	Text	C# Jump Start - Advanced Concepts	Microsoft
C#	Video	Top 20 C# Questions	Microsoft
C#	Text	C# Tutorial	tutorialspoint
Python	Text	Introduction to Financial Python	QuantConnect
Python	Text/Video	Introduction to Python	Google
Python	Interactive	Code Academy - Python	Code Academy
Python	Text	Python Pandas Tutorial	tutorialspoint

Enroll in Bootcamp

Bootcamp is an online coding experience where QuantConnect team and community members teach you to write your first algorithms. Bootcamp gives you step-by-step instructions on practical and beginner-friendly topics, so it's a great

way to learn LEAN. The lessons cover many topics that you use as you write your own algorithms, including universe selection , indicators , and consolidators . To get started, check out the course library .

Example Algorithm

The following video demonstrates how to implement an algorithm that buys and holds an S&P 500 index ETF:

```
# Import all the functionality you need to run algorithms
from AlgorithmImports import *

# Define a trading algorithm that is a subclass of QCAlgorithm
class MyAlgorithm(QCAlgorithm):
    # Define an Initialize method.
    # This method is the entry point of your algorithm where you define a series of settings.
    # LEAN only calls this method one time, at the start of your algorithm.
    def Initialize(self) -> None:
        # Set start and end dates
        self.SetStartDate(2018, 1, 1)
        self.SetEndDate(2022, 6, 1)
        # Set the starting cash balance to $100,000 USD
        self.SetCash(100000)
        # Add data for the S&P500 index ETF
        self.AddEquity("SPY")

    # Define an OnData method.
    # This method receives all the data you subscribe to in discrete time slices.
    # It's where you make trading decisions.
    def OnData(self, slice: Slice) -> None:
        # Allocate 100% of the portfolio to SPY
        if not self.Portfolio.Invested:
            self.SetHoldings("SPY", 1)
```

Key Concepts

Algorithm Engine

Introduction

LEAN Engine is an open-source algorithmic trading engine built for easy strategy research, backtesting, and live trading. We integrate with common data providers and brokerages, so you can quickly deploy algorithmic trading strategies. The core of the LEAN Engine is written in C#, but it operates seamlessly on Linux, Mac and Windows operating systems. To use it, you can write algorithms in Python 3.8 or C#. QuantConnect maintains the LEAN project and uses it to drive the web-based algorithmic trading platform on the website.

Since LEAN is open-source, you aren't locked-in to the QuantConnect platform. You can run LEAN with your own infrastructure, data, and brokerage connections. If you use QuantConnect, our team of engineers manage the infrastructure, you can run the latest version of LEAN with all of its brokerage connections, and you can utilize the datasets in the Dataset Market .

Your Algorithm and LEAN

To create a trading algorithm with LEAN, define a subclass of the `QCAlgorithm` class. LEAN loads your algorithm into the engine during the building and compilation process. LEAN runs an algorithm manager that synchronizes the data your algorithm requests, injects the data into your algorithm so you can place trades, processes your orders, and then updates your algorithm state.

The LEAN engine manages your portfolio and data feeds, so you focus on your algorithm strategy and execution. We automatically provide basic portfolio management and reality modeling underneath the hood. The `QCAlgorithm` class provides some key helper properties for you to use, including the Security Manager, Portfolio Manager, Transactions Manager, Notification Manager, and Scheduling Manager. The class also has hundreds of helper methods to make the API easy to use.

The Securities property is a dictionary of `Security` objects. Each asset (Equity, Forex pair, etc) in your algorithm has a `Security` object. All the models for a security live on these objects. For example, `self.Securities["IBM"].FeeModel` and `self.Securities["IBM"].Price` return the fee model and price of IBM, respectively.

The Portfolio is a dictionary of `SecurityHolding` objects. These classes track the profits, losses, fees, and quantity of individual portfolio holdings. For example, `self.Portfolio["IBM"].LastTradeProfit` returns the profit of your last IBM trade.

Other helpers like Transactions , Schedule , Notify , and Universe have their own helper methods.

```
class QCAlgorithm:
    Securities   # Array of Security objects.
    Portfolio    # Array of SecurityHolding objects
    Transactions # Transactions helper
    Schedule     # Scheduling helper
    Notify       # Email, SMS helper
    Universe     # Universe helper

    # Set up Requested Data, Cash, Time Period.
    def Initialize(self) -> None:

    # Other Event Handlers
    def OnData(self, slice: Slice) -> None:
    def OnEndOfDay(self, symbol: Symbol) -> None:
    def OnEndOfAlgorithm(self) -> None:

    # Indicator Helpers
    def SMA(self, symbol: Symbol, period: int) -> SimpleMovingAverage:
```

Threads in LEAN

LEAN is multi-threaded and attempts to consume as much CPU as possible to perform given work as quickly as possible. The analysis engine loads data parallel and synchronizes it to generate the requested security information.

The client algorithm is plugged into LEAN, and has its events triggered synchronously in backtesting. The primary bottle neck to LEAN execution is executing client code.

In live trading, most events are synchronous as in backtesting, however order events are triggered immediately from the brokerage thread (asynchronously) to ensure the lowest latency. For example; a strategy using hourly data, with a limit order could fill between data bars. This fill event is triggered when it occurs in live trading.

We recommend using thread safe collections when possible, or locks around collections to ensure they are thread-safe.

Batch vs Stream Analysis

Backtesting platforms come in two general varieties, batch processing and event streaming. Batch processing backtesting is much simpler. It loads all data into an array and passes it to your algorithm for analysis. Because your algorithm has access to future data points, it is easy to introduce look-ahead bias . Most home-grown analysis tools are batch systems.

QuantConnect/LEAN is a streaming analysis system. In live trading, your algorithm receives data points one after another over time. QuantConnect models this in backtesting, streaming data to your algorithm in fast-forward mode. Because of this, you can't access price data beyond the Time Frontier . Although streaming analysis is slightly trickier to understand, it allows your algorithm to seamlessly work in backtests and live trading with no code changes.

Event Flow

When you deploy an algorithm, LEAN first calls the Initialize method. In live mode, the engine loads your holdings and open orders from your brokerage account to add data subscriptions and populate the `Securities` , `Portfolio` , and `Transactions` objects. LEAN receives the data from the subscriptions, synchronizes the data to create a timeslice , and then performs the following steps. Your algorithm can spend up to 10 minutes on each timeslice unless you call the Train method.

1. If it's a backtest, check if there are Scheduled Events in the past that didn't fire because there was no data between the previous slice and the current slice. LEAN automatically creates a Scheduled Event to call the

`OnEndOfDay` method at the end of each day.

In live mode, Scheduled Events occur in a separate thread from the algorithm manager, so they run at the correct time.

2. Update the algorithm time.
3. Update the `CurrentSlice` .
4. Cancel all open orders for securities that changed their ticker .
5. Add a `Security` object to the `Securities` collection for each new security in the universe.
6. Update the `Security` objects with the latest data.
7. Update the `Cash` objects in the CashBook with the latest data.
8. Process fill models for non-market orders.
9. Submit market on open orders to liquidate Equity Option contracts if the underlying Equity has a split warning .
10. Process margin calls .
11. If it's time to settle unsettled cash , perform settlement.
12. Call the OnSecuritiesChanged method with the latest security changes.
13. Apply dividends to the portfolio.
14. For securities that have a split warning, update their portfolio holdings and adjust their open orders to account for the split.
15. Update consolidators with the latest data.
16. Pass the `Slice` to the `OnData` method.
17. Perform universe selection .
18. Pass the `Slice` to the `Update` method of each Alpha model .
19. Pass the `Insight` objects from the Alpha model to the Portfolio Construction model .
20. Pass the `PortfolioTarget` objects from the Portfolio Construction model to each Risk Management model .
21. Pass the risk-adjusted `PortfolioTarget` objects from the Risk Management models to the Execution model .

When your algorithm stops executing, LEAN calls the `OnEndOfAlgorithm` method .

Python Support

The LEAN engine is written in C#, but you can create algorithms in C# or Python. If you program in Python, LEAN uses Python.Net to bridge between the C# engine and your algorithm. As a result of the bridge, `QCAlgorithm` class members are in camel case, not snake case. It can be slow to move from Python to C#. If you access C# objects in your Python algorithm, it's fastest to only access them once and save a reference if you need to access them again.

```py
# Do this:
security_holding = self.Portfolio[self.symbol]
avg_price = security_holding.AveragePrice
quantity = security_holding.Quantity

# Avoid this:
avg_price = self.Portfolio[self.symbol].AveragePrice
quantity = self.Portfolio[self.symbol].Quantity
```

Key Concepts

Multi-Asset Modeling

Introduction

We designed LEAN as a multi-asset platform with out-of-the-box support for multiple securities, so you can model complex portfolios like hedge funds.

Asset Portfolio

The portfolio manages the individual securities it contains. It tracks the cost of holding each security. It aggregates the performance of the individual securities in the portfolio to produce statistics like net profit and drawdown . The portfolio also holds information about each currency in its cashbook.

Cashbooks

We designed LEAN to be a multi-currency platform. LEAN can trade Forex, Cryptocurrencies, and other assets that are quoted in other currencies. A benefit of supporting multiple currencies is that as we add new asset classes from new countries, LEAN is already prepared to transact in those assets by using their quote currency. For instance, we added the India Equity market, which quotes assets in the INR currency.

The portfolio manages your currencies in its cashbook, which models the cash as a ledger of transactions. When you buy assets, LEAN uses the currencies in your cashbook to purchase the asset and pay the transaction fees. For more information about the cashbook, see Cashbook .

Buying Power

We model the margin requirements of each asset and reflect that in the buying power available to the algorithm. We source Futures margins from CME SPAN margins. Equity margin is 2x for standard margin accounts and 4x intraday for Pattern Day Trading accounts. For more information about buying power modeling, see Buying Power .

Key Concepts

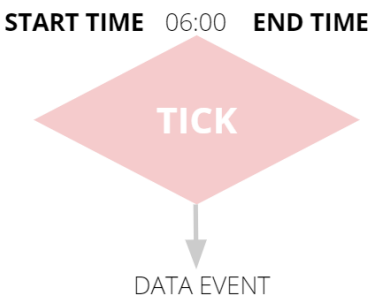
Time Modeling

Time Modeling

Periods

Introduction

Data comes in two different "shapes" according to the time period it covers: point values or period values. In QuantConnect, ticks are point values, and bars are a period values. These data formats have different properties, which control when LEAN emits them into your algorithm.



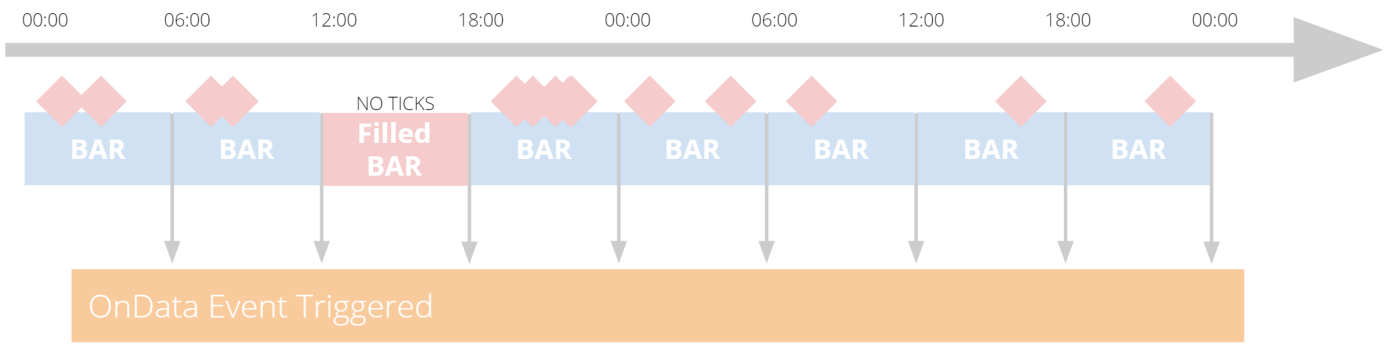
Start and End Time

Bars have a start and end time that represent the time period of which the bar aggregates data. LEAN passes the bar to your algorithm at the end time so that you don't receive the bar before it was actually available. Other data providers commonly timestamp their bars to the start time of the bar. This can cause one-off errors when you import the data into QC or compare indicator values across different platforms.

Period Values

Bars aggregate data across a period of time into a single object. We make this easy for you by pre-aggregating billions of raw trade-ticks into `TradeBar` objects and quote-ticks into `QuoteBar` objects.

We don't know the close of a bar until the start of the next bar, which can sometimes be confusing. For example, a price bar for Friday will include all the ticks from Friday 00:00:00 to Friday 23:59:59.99999, but LEAN will emit it to your algorithm on Saturday at midnight. As a result, if you analyze the Friday data and then place an order, LEAN sends the order to your brokerage on Saturday.



When there are no ticks during a period, LEAN emits the previous bar. This is the default behavior for bar data and it's referred to as "filling the data forward". You can enable and disable this setting when you create the security subscription .

We provide bar data in second, minute, hour, and daily bar formats. To create other periods of bars, see Consolidating Data .

Daily Periods

LEAN emits daily bars at midnight. To be notified when a security has finished trading for the day, add an `OnEndOfDay` method to your algorithm. LEAN calls this method for each security every day.

```
PY
def OnEndOfDay(self, symbol: Symbol) -> None:
    pass
```

Point Values

Point values have no period because they occur at a singular point in time. Examples of point data includes ticks, open interest, and news releases. Tick data represents a single trade or quote in the market. It's a discrete event with no period, so the `Time` and `EndTime` properties are the same. LEAN emits ticks as soon as they arrive and doesn't fill them forward.

00:00
06:00
12:00
18:00
00:00
06:00
12:00
18:00
00:00
T
OnData Event Triggered

Time Modeling

Timeslices

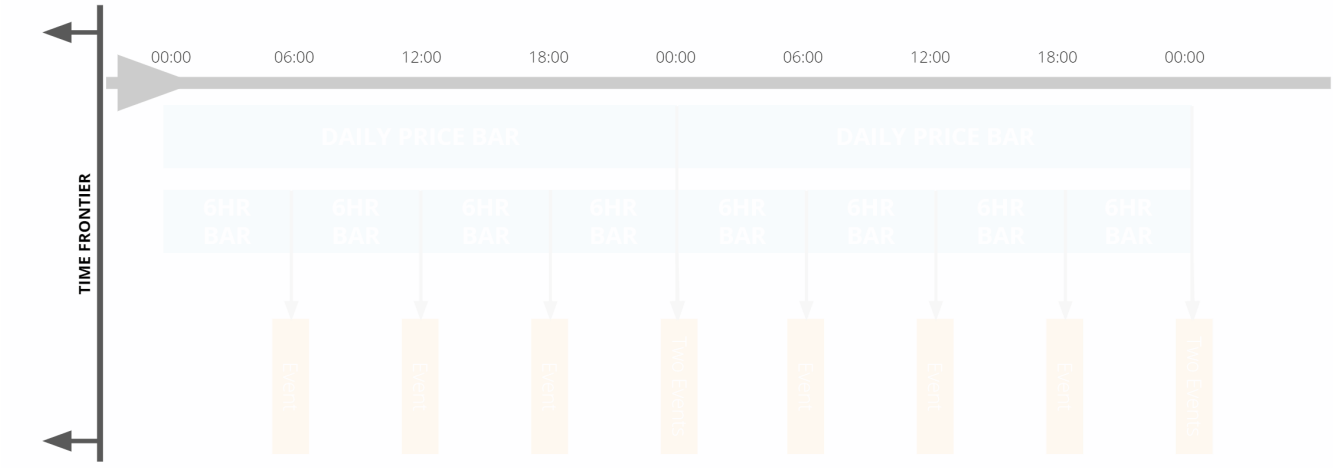
Introduction

The core technology behind QuantConnect algorithmic trading is an event-based, streaming analysis system called LEAN . LEAN attempts to model the stream of time as accurately as possible, presenting data ("events") to your algorithms in the order it arrives, as you would experience in reality.

All QuantConnect algorithms have this time-stream baked in as the primary event handler, `OnData` . The `Slice` object this method receives represents all of the data at a moment of time, a **time-slice** . No matter what data you request, you receive it in the order created according to simulated algorithm time. By only letting your algorithm see the present and past moments, we can prevent the most common quantitative-analysis error, look-ahead bias .

Time Frontier

If you request data for multiple securities and multiple resolutions, it can create a situation where one of your data subscriptions is ready to emit, but another subscription with a longer period is still be constructing its bar. Furthermore, if you request multiple datasets that have different time zones, your algorithm will receive the daily bars of each dataset at midnight in the respective time zone. To coordinate the data in these situations, we use the `EndTime` of each data point to signal when LEAN should transmit it to your algorithm.



Once your algorithm reaches the `EndTime` of a data point, LEAN sends the data to your `OnData` method. For bar data, this is the beginning of the next period. To avoid look-ahead bias, you can only access data from before this Time Frontier. The `Time` property of your algorithm is always equal to the Time Frontier.

Properties

`Slice` objects have the following properties:

Get Time Slices

To get the current `Slice` object, define an `OnData` method or use the `CurrentSlice` property of your algorithm. The

`Slice` contains all the data for a given moment in time. The `TradeBars` and `QuoteBars` properties are `Symbol` /string indexed dictionaries. The `Ticks` property is a list of ticks for that moment of time, indexed by the `Symbol` . To check which data formats are available for each asset class, see the Data Formats page in the Asset Classes chapter.

The `Slice` object gives you the following ways to access your data:

- Indexing the `Slice` , which returns a dynamic object of your type.

  ```
  def OnData(self, slice: Slice) -> None:
      data = slice[self.symbol]
  ```

  With minute and second resolution data, the dynamic type is `TradeBar` for Equities and `QuoteBar` for other asset classes.

- Indexing the static properties, which returns the type you specify.

  ```
  def OnData(self, slice: Slice) -> None:
      trade_bar = slice.Bars[self.symbol]
      quote_bar = slice.QuoteBars[self.symbol]
  ```

Check if the `Slice` contains the data you're looking for before you index it. If there is little trading, or you are in the same time loop as when you added the security, it may not have any data. Even if you enabled fill-forward for a security subscription, you should check if the data exists in the dictionary before you try to index it. To check if the `Slice` contains for a security, call the `ContainsKey` method. Note: if the `Slice` object doesn't contain any market data but it contains auxiliary data, the `slice.ContainsKey(symbol)` method can return true while `slice[symbol]` returns `None` .

```
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.symbol) and slice[self.symbol] is not None:
        data = slice[self.symbol]
```

Time Modeling

Time Zones

Introduction

LEAN is an international trading engine, so it can operate across time zones. The exchange, your algorithm, and the data in your algorithm can all have different time zones.

Exchange Time Zone

Exchanges are located in different areas, so they can have different time zones. To get the time zone of the exchange that a `Security` trades on, use its `Exchange.Hours.TimeZone` property.

```
PY
time_zone = self.Securities[self.symbol].Exchange.Hours.TimeZone
```

Algorithm Time Zone

LEAN supports international trading across multiple time zones and markets, so it needs a reference time zone for the algorithm to set the `Time` . The default time zone is Eastern Time (ET), which is UTC-4 in summer and UTC-5 in winter. To set a different time zone, call the `SetTimeZone` method. This method accepts either a string following the IANA Time Zone database convention or a NodaTime .DateTimeZone object. If you pass a string, the method converts it to a `NodaTime.DateTimeZone` object. The `TimeZones` class provides the following helper attributes to create `NodaTime.DateTimeZone` objects:

```
PY
self.SetTimeZone("Europe/London")
self.SetTimeZone(TimeZones.Chicago)
```

To get the time zone of your algorithm, use the `TimeZone` property.

```
PY
time_zone = self.TimeZone
```

To get the algorithm time in Coordinated Universal Time (UTC), use the `UtcTime` property.

```
PY
utc_time = self.UtcTime
```

The `Time` and `UtcTime` objects have no time zone. LEAN maintains their state to be consistent.

Data Time Zone

Datasets can have different time zones because the time zone is up to the data provider, where they are located, and where they collect the data. LEAN tracks the time zone of each dataset so that you receive the correct data at the right

point in time in your algorithm. If you have multiple datasets in your algorithm from different time zones, LEAN synchronizes them to your algorithm time. Furthermore, if you set up time period consolidators , LEAN consolidates the data based on the data time zone.

To get the time zone of a dataset, view the listing page in the Dataset Market or call the `GetDataTimeZone` method.

```
time_zone = self.MarketHoursDatabase.GetDataTimeZone(Market.USA, self.symbol, SecurityType.Equity)
```

Key Concepts

Security Identifiers

Introduction

We implemented `Symbol` objects in the LEAN open-source project as a way to identify or "finger-print" tradable assets so that no further database look-up is required. All QuantConnect and LEAN Algorithm API methods use `Symbol` objects to identify assets.

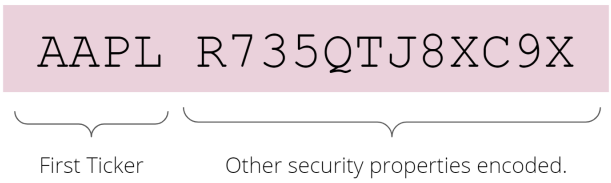
Encoding Symbols

`SecurityIdentifier` objects have several public properties to uniquely identify each asset. The following table describes these properties:

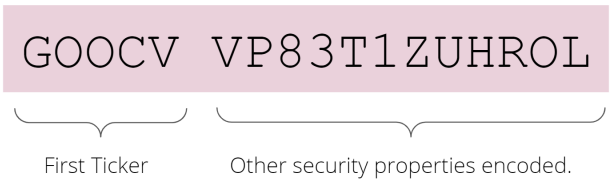
Property	Data Type	Description
`Market`	`string`	The market in which the asset trades
`SecurityType`	`SecurityType`	The asset class
`OptionStyle`	`OptionStyle`	American or European Option style
`OptionRight`	`OptionRight`	Call or put Option type
`StrikePrice`	`decimal`	The Option contract strike price
`Date`	`datetime`	Earliest listing date for Equities; expiry for Futures and Options; December 30, 1899 for continuous Futures contracts and Indices.
`HasUnderlying`	`bool`	A flag to represent if its a derivative asset with another underlying asset

We encode the preceding properties to create `Symbol` objects. We do our best to hide the details of this process from your algorithm, but you'll occasionally see it come through as an encoded hash like `AAPL R735QTJ8XC9X` . The first half of the encoded string represents the first ticker AAPL was listed under. The other letters at the end of the string represent the serialized form of the preceding `SecurityIdentifier` properties. You may also see an encoded has like `AAPL XL7X5I241SO6|AAPL R735QTJ8XC9X` for a derivative contract. In this case, the part before the `|` is contract hash and the part after the `|` is the underlying hash. This serialization approach lets LEAN assign a short, unique string to octillions of different security objects.

Apple Inc. (same name since listed).



Google Class-C Shares (renamed to GOOG).



LEAN assumes the ticker you pass to the AddEquity method is the current ticker of the Equity asset. To access this ticker, use the `Value` property of the `Symbol` object.

```py
self.symbol = self.AddEquity("GOOG").Symbol
self.Debug(self.symbol.ID.Value) # Prints "GOOCV"
self.Debug(self.symbol.Value)    # Prints "GOOG"
```

If you create the security subscription with a universe selection function, the `Value` property of the `Symbol` object is the point-in-time ticker.

Decoding Symbols

When a `SecurityIdentifier` is serialized to a string, it looks something like `SPY R735QTJ8XC9X` . This two-part string is a base64 encoded set of data. Encoding all of the properties into a short format allows dense communication without requiring a third-party list or look-up. Most of the time, you will not need to work with these encoded strings. However, to deserialize the string into a `Symbol` object, call the `Symbol` method.

```py
symbol = self.Symbol("GOOCV VP83T1ZUHROL")
symbol.ID.Market     # => "USA"
symbol.SecurityType  # => SecurityType.Equity
symbol.Value         # => "GOOCV"
```

The `Market` property distinguishes between tickers that have the same string value but represent different underlying assets. A prime example of this is the various market makers who have different prices for EURUSD. We store this data separately and as they have different fill prices, we treat the execution venues as different markets.

Tickers

Tickers are a string shortcode representation for an asset. Some examples of popular tickers include "AAPL" for Apple Corporation and "IBM" for International Business Machines Corporation. These tickers often change when the company rebrands or they undergo a merger or reverse merger.

The ticker of an asset is not the same as the `Symbol` . `Symbol` objects are permanent and track the underlying entity. When a company rebrands or changes its name, the `Symbol` object remains constant, giving algorithms a way to reliably track assets over time.

Tickers are also often reused by different brokerages. For example Coinbase, a leading US Crypto Brokerage lists the "BTCUSD" ticker for trading. Its competitor, Bitfinex, also lists "BTCUSD". You can trade both tickers with LEAN. `Symbol` objects allow LEAN to identify which market you reference in your algorithms.

To create a `Symbol` object for a point-in-time ticker, call the `GenerateEquity` method to create the security identifier and then call the `Symbol` constructor. For example, Heliogen, Inc. changed their ticker from ATHN to HLGN on December 31, 2021. To convert the ATHN ticker to the Equity `Symbol` , type:

```
PY
ticker = "ATHN"
security_id = SecurityIdentifier.GenerateEquity(ticker, Market.USA, mappingResolveDate=datetime(2021, 12,
1))
symbol = Symbol(security_id, ticker)
```

In the preceding code snippet, the `mappingResolveDate` must be a date when the point-in-time ticker was trading.

Symbol Cache

To make using the API easier, we have built the **Symbol Cache** technology, which accepts strings and tries to guess which `Symbol` object you might mean. With the Symbol Cache, many methods can accept a string such as "IBM" instead of a complete `Symbol` object. We highly recommend you don't rely on this technology and instead save explicit references to your securities when you need them.

```
PY
# Example 1: Relying On Symbol Cache:
self.AddEquity("IBM")         # Add by IBM string ticker, save reference to Symbol Cache.
self.MarketOrder("IBM", 100)  # Determine refering to IBM Equity from Symbol Cache.
self.History("AAPL", 14)      # Makes a guess referring to AAPL Equity.

# Example 2: Correctly Using Symbols:
self.ibm = self.AddEquity("IBM").Symbol   # Add IBM Equity string ticker, save Symbol.
self.MarketOrder(self.ibm, 100)           # Use IBM Symbol in future API calls.

self.aapl = Symbol.Create("AAPL", SecurityType.Equity, Market.USA)
self.History(self.aapl, 14)
```

Industry Standard Identifiers

You can convert industry-standard security identifiers like CUSIP, FIGI, ISIN, and SEDOL to `Symbol` objects and you can convert `Symbol` objects to industry-standard security identifiers.

CUSIP

To convert a Committee on Uniform Securities Identification Procedures (CUSIP) number to a `Symbol` or a `Symbol` to a CUSIP number, call the `CUSIP` method.

```
PY
symbol = self.CUSIP("03783310") # AAPL Symbol
cusip = self.CUSIP(symbol) # AAPL CUSIP (03783310)
```

FIGI

To convert a Financial Instrument Global Identifier (FIGI) to a `Symbol` or a `Symbol` to a FIGI, call the `CompositeFIGI` method.

```
symbol = self.CompositeFIGI("BBG000B9XRY4") # AAPL Symbol
figi = self.CompositeFIGI(symbol) # AAPL FIGI (BBG000B9XRY4)
```

ISIN

To convert an International Securities Identification Number (ISIN) to a `Symbol` or a `Symbol` to an ISIN, call the `ISIN` method.

```
symbol = self.ISIN("US0378331005") # AAPL Symbol
isin = self.ISIN(symbol) # AAPL ISIN (US0378331005)
```

SEDOL

To convert a Stock Exchange Daily Official List (SEDOL) number to a `Symbol` or a `Symbol` to a SEDOL number, call the `SEDOL` method.

```
symbol = self.SEDOL("2046251") # AAPL Symbol
sedol = self.SEDOL(symbol) # AAPL SEDOL (2046251)
```

Key Concepts

Event Handlers

Introduction

Event handlers are called during an algorithm execution to pass information to your strategy. Most of the events are not needed for simple strategies, but can be helpful for debugging issues in more complex algorithms.

Data Events

```py
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.symbol):
        my_data = slice[self.symbol]
```

The `OnData` method is the primary event handler for receiving financial data events to your algorithm. It is triggered sequentially at the point in time the data is available; in backtesting and live. For daily data this means the event is processed at midnight in backtesting or 6am the following day in live trading.

All data for a given moment of time is grouped in a single event, including custom data types. This data is passed with the Slice object.

When fill-forward is enabled for your asset, the OnData event handler will be called regularly even if there was no new data. This is the default behavior.

In backtesting, if your algorithm takes a long time to process a slice, the following slice objects queue up and the next event triggers when your algorithm finishes processing the current slice. In live trading, if your algorithm takes longer to process the current slice than the time period that the slice spans, the next event triggers when your algorithm finishes processing the current slice, but the slice of the following event is the most recent slice. For example, say your algorithm consumes second resolution Crypto data but it takes your algorithm 3.5 seconds to process each slice. In backtesting, you'll get every slice. In live trading, if you deploy at 12:00:00 AM Coordinated Universal Time (UTC), you'll get the first slice at 12:00:01 AM UTC (spanning 12:00:00 AM UTC to 12:00:01 AM UTC) and you'll get the second slice at 12:00:04.5 AM UTC (roughly spanning 12:00:03 AM UTC to 12:00:04 AM UTC).

To perform intensive computation before the market opens, use a Scheduled Event or the Train method.

For more information on the `Slice` object and OnData event, see Handling Data .

Securities Changed Event

```
def OnSecuritiesChanged(self, changes: SecurityChanges) -> None:
    for security in changes.AddedSecurities:
        self.Debug(f"{self.Time}: Added {security.Symbol}")

    for security in changes.RemovedSecurities:
        self.Debug(f"{self.Time}: Removed {security.Symbol}")

        if security.Invested:
            self.Liquidate(security.Symbol, "Removed from Universe")
```

The `OnSecuritiesChanged` event notifies the algorithm when assets are added or removed from the universe. This can be due to changes in the Universe constituents , or an explicit call to the RemoveSecurity method.

The event is triggered immediately when the asset is removed from the universe; however the data feed for the asset may remain active if the algorithm has open orders.

For more information, see how to use Security Changed Events .

Order Events

```
def OnOrderEvent(self, orderEvent: OrderEvent) -> None:
    order = self.Transactions.GetOrderById(orderEvent.OrderId)
    if orderEvent.Status == OrderStatus.Filled:
        self.Debug(f"{self.Time}: {order.Type}: {orderEvent}")

def OnAssignmentOrderEvent(self, assignmentEvent: OrderEvent) -> None:
    self.Log(str(assignmentEvent))
```

The `OnOrderEvent` method notifies the algorithm of new orders, and changes in the order status such as fill events and cancelations. For options assignment events there is a dedicated event handler `OnAssignmentOrderEvent` .

In backtesting order events are triggered synchronously after the main data events. In live trading, order events are asynchronously as they occur. To avoid infinite loops, we recommend not to place orders in the OnOrderEvent.

For more information, see how to use Order Events .

Brokerage Events

```
def OnBrokerageMessage(self, message: BrokerageMessageEvent) -> None:
    if message.Type == BrokerageMessageType.Reconnect:
        self.Log(f"{self.Time}: {message.Type}: Message: {message.Message}")

def OnBrokerageDisconnect(self) -> None:
    self.Log("Brokerage disconnection detected")
```

The `OnBrokerageDisconnect` and `OnBrokerageReconnect` event handlers pass information to the algorithm about the brokerage connection status. This can be helpful for considering when to place a trade when a brokerage API is unreliable or under maintenance. The `OnBrokerageMessage` event handler provides information from the brokerages, including the disconnect and reconnect messages. Message content varies with each brokerage.

Brokerage events are triggered asynchronously in live trading, and are not created in backtesting.

Margin Call Events

```
def OnMarginCall(self, requests): -> List[SubmitOrderRequest]:
    # Modify order requests to choose what to liquidate.
    return requests

def OnMarginCallWarning()(self) -> None:
    self.Log("Margin call warning, 5% margin remaining")
```

The `OnMarginCallWarning` and `OnMarginCall` event handlers provide information and control over how to reduce algorithm leverage when performance is poor.

The `OnMarginCallWarning` method is called when there is less than 5% margin remaining, this gives you an opportunity to reduce leverage before an official margin call is performed by the brokerage. The `OnMarginCall` event handler is passed a list of `SubmitOrderRequest` objects which will be executed on exiting the method.

Margin events are called before the data events are processed.

The following demonstration processes suggested orders and modifies the qualities to liquidate more than necessary and prevent a second margin call. For more information, see how to handle Margin Calls .

Warmup Finished Event

```
def OnWarmupFinished(self) -> None:
    pass # Done warming up
```

The `OnWarmupFinished` event handler is triggered after initialization, and warm up is complete. This event signals that the initialization of the algorithm is complete.

For more information, see how to use Warm Up to prepare your algorithm.

End Of Algorithm Events

```
def OnEndOfAlgorithm(self) -> None:
    self.Debug("Algorithm done")
```

The `OnEndOfAlgorithm` event handler is when the algorithm has finished executing as its final step. This is helpful for performing final analysis, or saving algorithm state.

Event Flow

When you deploy an algorithm, LEAN first calls the Initialize method. In live mode, the engine loads your holdings and open orders from your brokerage account to add data subscriptions and populate the `Securities` , `Portfolio` , and `Transactions` objects. LEAN receives the data from the subscriptions, synchronizes the data to create a timeslice , and then performs the following steps. Your algorithm can spend up to 10 minutes on each timeslice unless you call the Train method.

1. If it's a backtest, check if there are Scheduled Events in the past that didn't fire because there was no data between the previous slice and the current slice. LEAN automatically creates a Scheduled Event to call the `OnEndOfDay` method at the end of each day.

In live mode, Scheduled Events occur in a separate thread from the algorithm manager, so they run at the correct time.

2. Update the algorithm time.
3. Update the `CurrentSlice` .
4. Cancel all open orders for securities that changed their ticker .
5. Add a `Security` object to the `Securities` collection for each new security in the universe.
6. Update the `Security` objects with the latest data.
7. Update the `Cash` objects in the CashBook with the latest data.
8. Process fill models for non-market orders.
9. Submit market on open orders to liquidate Equity Option contracts if the underlying Equity has a split warning .
10. Process margin calls .
11. If it's time to settle unsettled cash , perform settlement.
12. Call the OnSecuritiesChanged method with the latest security changes.
13. Apply dividends to the portfolio.
14. For securities that have a split warning, update their portfolio holdings and adjust their open orders to account for the split.
15. Update consolidators with the latest data.
16. Pass the `Slice` to the `OnData` method.
17. Perform universe selection .
18. Pass the `Slice` to the `Update` method of each Alpha model .
19. Pass the `Insight` objects from the Alpha model to the Portfolio Construction model .
20. Pass the `PortfolioTarget` objects from the Portfolio Construction model to each Risk Management model .
21. Pass the risk-adjusted `PortfolioTarget` objects from the Risk Management models to the Execution model .

When your algorithm stops executing, LEAN calls the `OnEndOfAlgorithm` method .

Key Concepts

Python and LEAN

Introduction

The LEAN open-source package powers the core functionality of QuantConnect. Strategies plugin to LEAN to request data, and process trades. LEAN is written in C# and accessed in python with a defined API. Most of these interactions are using native python types, with a few key exceptions. To comfortably use LEAN you should learn to recognize error messages when you hit a non-native API, and how to convert back to a native type.

Enumerable Types

When LEAN returns an enumerable, the object should be converted to a `list()` to be easily read by python classes. Popular API methods which return an enumerable are shown below:

Historical Data

QuantConnect also provides methods to request historical data that return a pandas dataframe, or an enumerable. If loading a large amount of data using an enumerable streams the results reducing RAM usage.

The enumerable types might look like the `MemoizingEnumerable` that is a raw C# type accessible in python.

```
qb = QuantBook()
start = datetime(2023, 2, 1)
end = datetime(2023, 2, 4)
ibm = qb.AddEquity('IBM', Resolution.Minute)
bars = qb.History[TradeBar](ibm.Symbol, start, end, Resolution.Minute)
bars
```

✓ 0.3s Python

```
<QuantConnect.Util.MemoizingEnumerable[TradeBar] object at 0x7fd5313209c0>
```

You can directly enumerate this object with standard python `for` loops, or to convert it to a native type, wrap the object in the `list()` :

```
# Loop over enumerable
import numpy as np
close = []
for bar in bars:
    close.append( bar.Close )
print( np.mean(close) )

# Manipulate as a list
barList =  list(bars)
len(barList)
```

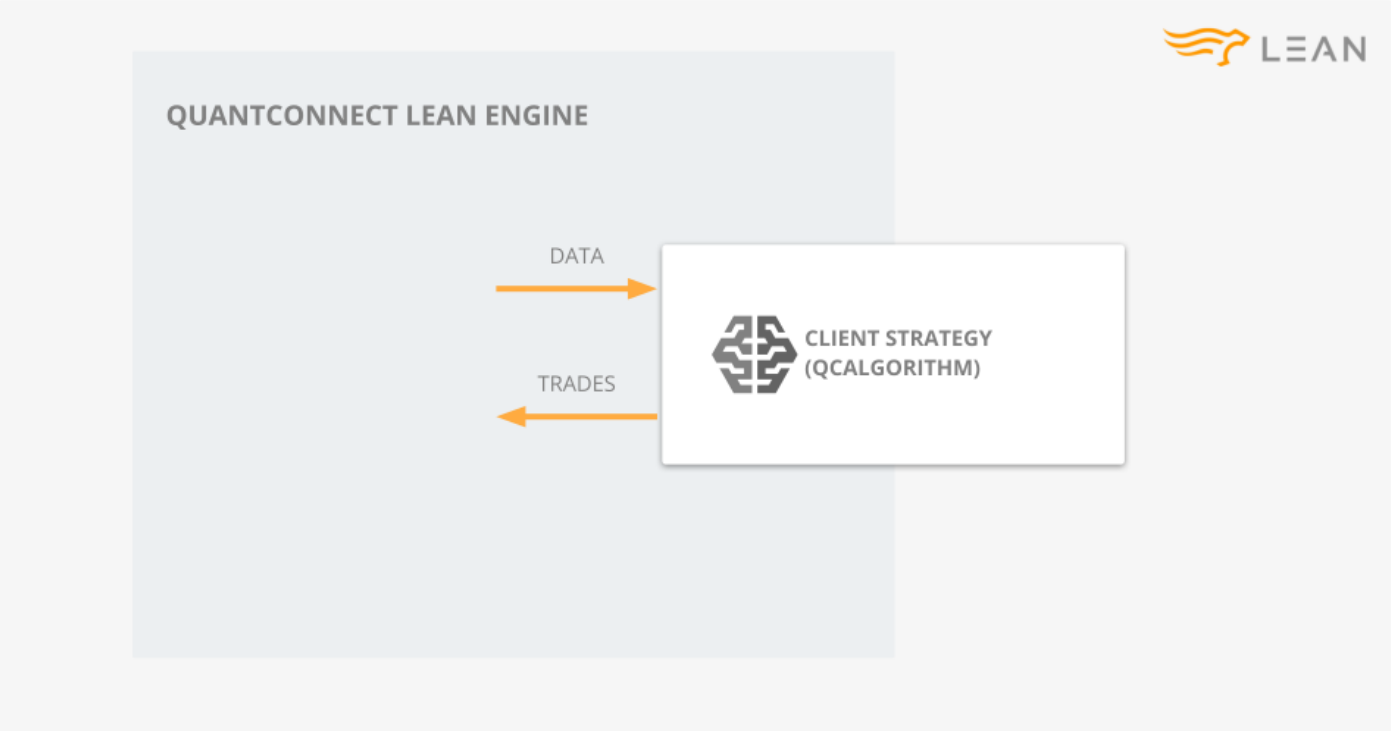
✓ 0.8s Python

```
135.29654786324787

1170
```

Underlying Plugin Architecture

Your strategy code implements the QCAlgorithm class, which can communicate with the LEAN Engine. These method

calls are done at the same speed as cython, directly invoking calls at the C layer of python. Your algorithm is imported as syntax checked, interpreted python.



Key Concepts

Glossary

Introduction

This page defines terms in QuantConnect products and documentation.

alpha

The quantity of an algorithm's returns that aren't explained by its underlying benchmark.

annual standard deviation

A staticial measure that describes the dispersion of annual returns relative to the mean annual return. It's the square root of the annual variance.

annual variance

A staticial measure that describes the dispersion of annual returns relative to the mean annual return.

average loss

The average rate of return for unprofitable trades.

average win

The average rate of return for profitable trades.

beta

The scale and direction of an algorithm's returns relative to movements in the underlying benchmark.

capacity

The maximum amount of money an algorithm can trade before its performance degrades from market impact.

compounding annual return

The annual percentage return that would be required to grow a portfolio from its starting value to its ending value.

day trade

Buying and selling the same asset within one trading day.

drawdown

The largest peak to trough decline in an algorithm's equity curve.

equity

The total portfolio value if all of the holdings were sold at current market rates.

expectancy

The expected return per trade.

holdings

The absolute sum of the items in the portfolio.

information ratio

The amount of excess return from the risk-free rate per unit of systematic risk.

intrinsic value

(Call Option) The price of an asset minus the strike price if the price is above the strike price, otherwise zero.

(Put Option) The strike price minus the price of an asset if the price is below the strike price, otherwise zero.

look-ahead bias

The practice of making decisions using information that would not be available until some time in the future.

loss rate

The proportion of trades that were not profitable.

lowest capacity asset

The asset an algorithm traded that has the lowest capacity.

net profit

(Percent) The rate of return across the entire trading period.

(Dollar-value) The dollar-value return across the entire trading period.

out-of-the-money amount

(Call Option) The strike price minus the price of an asset if the price is below the strike price, otherwise zero.

(Put Option) The price of an asset minus the strike price if the price is above the strike price, otherwise zero.

payoff

The profit or loss that an Option buyer or seller makes from a trade.

pattern day trader

A trader who executes four or more day trades in the US Equity market within five business days.

Probabilistic Sharpe ratio

The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).

profit-loss ratio

The ratio of the average win rate to the average loss rate.

return

The rate of return across the entire trading period.

Sharpe ratio

A measure of the risk-adjusted return, developed by William Sharpe.

total fees

The total quantity of fees paid for all the transactions.

total net profit

The rate of return across the entire trading period.

total trades

The number of orders that were filled or partially filled.

tracking error

A measure of how closely a portfolio follows the index to which it is benchmarked. A tracking error of 0 is a perfect match.

Treynor ratio

A measurement of the returns earned in an algorithm in excess of the risk-free rate per unit of benchmark risk, developed by Jack Treynor.

unrealized

The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.

volume

The total value of assets traded for all of an algorithm's transactions.

win rate

The proportion of trades that were profitable after transaction fees.

Key Concepts

Research Guide

Introduction

We aim to teach and inspire our community to create high-performing algorithmic trading strategies. We measure our success by the profits community members create through their live trading. As such, we try to build the best quantitative research techniques possible into the product to encourage a robust research process.

Hypothesis-Driven Research

We recommend you develop an algorithmic trading strategy based on a central hypothesis. You should develop an algorithm hypothesis at the start of your research and spend the remaining time exploring how to test your theory. If you find yourself deviating from your core theory or introducing code that isn't based around that hypothesis, you should stop and go back to thesis development.

Wang et al. (2014) illustrate the danger of creating your hypothesis based on test results. In their research, they examined the earnings yield factor in the technology sector over time. During 1998-1999, before the tech bubble burst, the factor was unprofitable. If you saw the results and then decided to bet against the factor during 2000-2002, you would have lost a lot of money because the factor performed extremely well during that time.

Hypothesis development is somewhat of an art and requires creativity and great observation skills. It is one of the most powerful skills a quant can learn. We recommend that an algorithm hypothesis follow the pattern of cause and effect. Your aim should be to express your strategy in the following sentence:

> *A change in {cause} leads to an {effect}.*

To search for inspiration, consider causes from your own experience, intuition, or the media. Generally, causes of financial market movements fall into the following categories:

- Human psychology
- Real-world events/fundamentals
- Invisible financial actions

Consider the following examples:

Cause	*leads to*	Effect
Share class stocks are the same company, so any price divergence is irrational...		A perfect pairs trade. Since they are the same company, the price will revert.
New stock addition to the S&P500 Index causes fund managers to buy up stock...		An increase in the price of the new asset in the universe from buying pressure.
Increase in sunshine-hours increases the production of oranges...		An increase in the supply of oranges, decreasing the price of Orange Juice Futures.
Allegations of fraud by the CEO causes investor faith in the stock to fall...		A collapse of stock prices for the company as people panic.
FDA approval of a new drug opens up new markets for the pharmaceutical company...		A jump in stock prices for the company.
Increasing federal interest rates restrict lending from banks, raising interest rates...		Restricted REIT leverage and lower REIT ETF returns.

There are millions of potential alpha strategies to explore, each of them a candidate for an algorithm. Once you have chosen a strategy, we recommend exploring it for no more than 8-32 hours, depending on your coding ability.

Data Mining Driven Research

An alternative view is to follow any statistical anomaly without explaining it. In this case, you can use statistical techniques to identify the discontinuities and eliminate them when their edge is gone. Apparently, Renaissance Technologies has data mining models like this.

Overfitting

Overfitting occurs when you fine-tune the parameters of an algorithm to fit the detail and noise of backtesting data to the extent that it negatively impacts the performance of the algorithm on new data. The problem is that the parameters don't necessarily apply to new data and thus negatively impact the algorithm's ability to generalize and trade well in all market conditions. The following table shows ways that overfitting can manifest itself:

Data Practice	Description
Data Dredging	Performing many statistical tests on data and only paying attention to those that come back with significant results.
Hyper-Tuning Parameters	Manually changing algorithm parameters to produce better results without altering the test data.
Overfit Regression Models	Regression, machine learning, or other statistical models with too many variables will likely introduce overfitting to an algorithm.
Stale Testing Data	Not changing the backtesting data set when testing the algorithm. Any improvements might not be able to be generalized to different datasets.

An algorithm that is dynamic and generalizes to new data is more valuable to funds and individual investors. It is more likely to survive across different market conditions and apply to new asset classes and markets.

If you have a collection of factors, you can backtest over a period of time to find the best-performing factors for the

time period. If you then narrow the collection of factors to just the best-performing ones and backtest over the same period, the backtest will show great results. However, if you take the same best-performing factors and backtest them on an out-of-sample dataset, the performance will almost always underperform the in-sample period. To avoid issues with overfitting, follow these guidelines:

- Use walk-forward optimization to train your models on historical data and test them on future data.
- Test your strategy with live paper trading.
- Test your model on different asset classes and markets.

Look-ahead Bias

Look-ahead bias occurs when you use information from the future to inform decisions in the present. An example of look-ahead bias is using financial statement data to make trading decisions at the end of the reporting period instead of when the financial statement data was released. Another example is using updated financial statement data before the updated figures were actually available. Wang et al. (2014) show that using the date of when the period ends instead of when the data is actually available can increase the performance of the earnings yield factor by 60%.

Another culprit of look-ahead bias is adjusted price data. Splits and reverse splits can improve liquidity or attract certain investors, causing the performance of the stock to be different than without the split or reverse split. Wang et al (2014) build a portfolio using the 25 lowest priced stocks in the S&P index based on adjusted and non-adjusted prices. The portfolio based on adjusted prices greatly outperformed the one with raw prices in terms of return and Sharpe ratio . In this case, if you were to analyze the low price factor with adjusted prices, it would lead you to believe the factor is very informative, but it would not perform well out-of-sample with raw data.

Look-ahead bias can also occur if you set the universe to assets that have performed well during the backtest period or initialize indicators with values that have performed well during the backtest period. To avoid issues with look-ahead bias, trade a dynamic universe of assets and use point-in-time data. If point-in-time data is not available for the dataset you use, apply a reporting lag. Since the backtesting environment is event-driven and you can't pass the time frontier, it naturally helps to reduce the risk of look-ahead bias.

Survivorship Bias

Survivorship bias occurs when you test a strategy using only the securities that have survived to the end of the testing period and omit securities that have been delisted. If you use the current constituents of an index and backtest over the past, you'll most likely outperform the index because many of the underperformers have dropped out of the index over time and outperformers have been added to the index. In this case, the current constituent universe would consist of only outperformers. This technique is a form of look-ahead bias because if you were trading the strategy in real-time throughout the backtest period, you would not know the current index constituents until today.

If you analyze a dataset that has survivorship bias, you can discover results that are opposite of the true results. For example, Wang et al. (2014) analyze the low volatility factor using two universes. The first universe was the current S&P 500 constituents and the second universe was the point-in-time constituents. When they used the point-in-time constituents, the low volatility quintile outperformed the high volatility quintile. However, when they used the current constituents, the high volatility quintile significantly outperformed the low volatility quintile.

To avoid issues with survivorship bias, trade a dynamic universe of assets and use the datasets in the Dataset Market . We thoroughly vet the datasets we add to the market to ensure they're free of survivorship bias.

Outliers

Outliers in a dataset can have large impacts on how models train. In some cases, you may leave outliers in a dataset because they can contain useful information. In other cases, you may want to transform the data to handle the outlier data points. There are several common methods to handle outliers.

Winsorization Method

The winsorization method removes outliers at the x% of the extremes. For example, if you winsorize at 1%, it removes the 1% of data points with the lowest value and the 1% of data points with the highest value. This threshold percentage is subjective, so it can result in overfitting.

IQR Method

The interquartile range method removes data points that fall outside of the interval $[Q_1 - k(Q_3 - Q_1),\ Q_3 + k(Q_3 - Q_1)]$ for some constant $k \geq 0$. This method can exclude up to 25% of observations on each side of the extremes. The interquartile range method doesn't work if the data is skewed or non-normal. Therefore, the disadvantage to this method is you need to review the factor distribution properties. If you need to, you can normalize the data with z-scores.

$$Z = \frac{x - \mu}{\sigma}$$

Factor Ranking Method

The factor ranking method ranks the data values instead of taking their raw values. With this method, you don't need to remove any outlier data points. This method transforms the data into a uniform distribution. After converting the data to a uniform distribution, Wang et al. (2014) perform an inverse normal transformation to make the factor values normally distributed. Wang et al. found this ranking technique outperforms the z-score transformation, suggesting that the distance between factor scores doesn't add useful information.

Key Concepts

Libraries

Introduction

Libraries (or packages) are third-party software that you can use in your projects. You can use many of the available open-source libraries to complement the classes and methods that you create. Libraries reduce your development time because it's faster to use a pre-built, open-source library than to write the functionality. Libraries can be used in backtesting, research, and live trading. The environments support various libraries for machine learning, plotting, and data processing. As members often request new libraries, we frequently add new libraries to the underlying docker image that runs the Lean engine.

This feature is primarily for Python algorithms as not all Python libraries are compatible with each other. We've bundled together different sets of libraries into distinct environments. To use the libraries of an environment, set the environment in your project and add the relevant `import` statement of a library at the top of your file.

Default Environment Libraries

The default environment supports the following libraries:

```
# Name                      Version
absl-py                     1.4.0
adagio                      0.2.4
aesara                      2.8.12
aiodns                      3.0.0
aiohttp                     3.8.4
aiohttp-cors                0.7.0
aiosignal                   1.3.1
alabaster                   0.7.13
ale-py                      0.7.4
alembic                     1.10.3
alpaca-trade-api            0.26
alphalens-reloaded          0.4.3
altair                      4.2.2
ansi2html                   1.8.0
antlr4-python3-runtime      4.11.1
anyio                       3.6.2
appdirs                     1.4.4
arch                        5.3.1
argon2-cffi                 21.3.0
argon2-cffi-bindings        21.2.0
arviz                       0.15.1
astropy                     5.2.1
asttokens                   2.2.1
astunparse                  1.6.3
async-timeout               4.0.2
asyncio-nats-client         0.11.5
attrs                       23.1.0
autograd                    1.5
autokeras                   1.1.0
autoray                     0.6.3
AutoROM                     0.4.2
AutoROM.accept-rom-license  0.6.1
ax-platform                 0.3.0
Babel                       2.12.1
backcall                    0.2.0
bayesian-optimization       1.4.2
beautifulsoup4              4.11.2
bleach                      6.0.0
blessed                     1.20.0
```

```
blis                          0.7.9
blosc2                        2.0.0
bokeh                         2.4.3
boltons                       23.0.0
botorch                       0.8.2
Bottleneck                    1.3.7
brotlipy                      0.7.0
cachetools                    5.3.0
captum                        0.6.0
catalogue                     2.0.8
catboost                      1.1.1
category-encoders             2.6.0
ccxt                          3.0.72
certifi                       2022.12.7
cffi                          1.15.1
chardet                       5.1.0
charset-normalizer            2.0.4
check-shapes                  1.0.0
click                         7.1.2
clikit                        0.6.2
cloudpickle                   2.2.1
cmaes                         0.9.1
cmdstanpy                     1.1.0
colorama                      0.4.6
colorcet                      3.0.1
colorful                      0.5.5
colorlog                      6.7.0
colorlover                    0.3.0
colour                        0.1.5
comm                          0.1.3
commonmark                    0.9.1
conda                         23.3.1
conda-content-trust           0.1.3
conda-package-handling        2.0.2
conda_package_streaming       0.7.0
confection                    0.0.4
cons                          0.4.5
contourpy                     1.0.7
convertdate                   2.4.0
copulae                       0.7.7
copulas                       0.8.0
cramjam                       2.6.2
crashtest                     0.3.1
creme                         0.6.1
cryptography                  38.0.4
cufflinks                     0.17.3
cvxopt                        1.3.0
cvxpy                         1.3.0
cycler                        0.11.0
cymem                         2.0.7
Cython                        0.29.33
darts                         0.23.1
dash                          2.9.3
dash-core-components          2.0.0
dash-cytoscape                0.3.0
dash-html-components          2.0.0
dash-table                    5.0.0
dask                          2023.4.0
deap                          1.3.3
debugpy                       1.5.1
decorator                     5.1.1
defusedxml                    0.7.1
Deprecated                    1.2.13
dgl                           1.0.1
dill                          0.3.6
dimod                         0.12.3
diskcache                     5.6.1
distlib                       0.3.6
distributed                   2021.4.1
dm-tree                       0.1.8
docutils                      0.19
DoubleML                      0.5.2
dtreeviz                      2.2.1
dtw-python                    1.3.0
dwave-cloud-client            0.10.3
dwave-drivers                 0.4.4
dwave-greedy                  0.3.0
dwave-hybrid                  0.6.9
dwave-inspector               0.3.0
dwave-inspectorapp            0.3.0
```

```
dwave-neal                    0.6.0
dwave-networkx                0.8.12
dwave-ocean-sdk               6.1.1
dwave-preprocessing           0.5.3
dwave-samplers                1.0.0
dwave-system                  1.18.0
dwave-tabu                    0.5.0
dwavebinarycsp                0.2.0
dx                            0.1.22
ecos                          2.0.12
EMD-signal                    1.4.0
empyrical                     0.5.5
empyrical-reloaded            0.5.9
en-core-web-md                3.5.0
en-core-web-sm                3.5.0
entrypoints                   0.4
ephem                         4.1.4
et-xmlfile                    1.1.0
etuples                       0.3.8
exceptiongroup                1.1.1
exchange-calendars            4.2.6
executing                     1.2.0
ezyrb                         1.3.0.post2304
fastai                        2.7.11
fastai2                       0.0.30
fastcore                      1.5.29
fastdownload                  0.0.7
fasteners                     0.18
fastjsonschema                2.16.3
fastparquet                   2023.2.0
fastprogress                  1.0.3
fasttext                      0.9.2
featuretools                  0.23.1
filelock                      3.12.0
findiff                       0.9.2
finrl                         0.3.1
FixedEffectModel              0.0.5
Flask                         2.0.3
flatbuffers                   23.3.3
fonttools                     4.39.3
fracdiff                      0.9.0
frozendict                    2.3.7
frozenlist                    1.3.3
fs                            2.4.16
fsspec                        2023.4.0
fst-pso                       1.8.1
fugue                         0.8.3
fugue-sql-antlr               0.1.6
future                        0.18.3
fuzzy-c-means                 1.6.3
FuzzyTM                       2.0.5
gast                          0.4.0
gensim                        4.3.0
gevent                        22.10.2
gluonts                       0.12.3
google-api-core               2.11.0
google-auth                   2.17.3
google-auth-oauthlib          0.4.6
google-pasta                  0.2.0
googleapis-common-protos      1.59.0
gpflow                        2.7.0
gplearn                       0.4.2
gpustat                       1.1
GPUtil                        1.4.0
gpytorch                      1.9.1
graphviz                      0.20.1
greenlet                      2.0.2
grpcio                        1.54.0
gym                           0.21.0
Gymnasium                     0.26.3
gymnasium-notices             0.0.1
h2o                           3.40.0.1
h5netcdf                      1.1.0
h5py                          3.8.0
hijri-converter               2.2.4
hmmlearn                      0.2.8
holidays                      0.23
holoviews                     1.15.4
homebase                      1.0.1
hopcroftkarp                  1.2.5
```

```
[illegible]                   [illegible]
html5lib                      1.1
httpstan                      4.9.1
hurst                         0.0.5
hvplot                        0.8.2
hyperopt                      0.2.7
idna                          3.4
iisignature                   0.24
ijson                         3.2.0.post0
imageio                       2.27.0
imagesize                     1.4.1
imbalanced-learn              0.10.1
importlib-metadata            4.13.0
importlib-resources           5.12.0
iniconfig                     2.0.0
injector                      0.20.1
interpret                     0.3.0
interpret-core                0.3.2
ipykernel                     6.22.0
ipython                       8.12.0
ipython-genutils              0.2.0
ipywidgets                    8.0.4
itsdangerous                  2.1.2
jax                           0.4.4
jaxlib                        0.4.4
jedi                          0.18.2
Jinja2                        3.1.2
joblib                        1.2.0
jqdatasdk                     1.8.11
json5                         0.9.11
jsonpatch                     1.32
jsonpointer                   2.0
jsonschema                    4.17.3
jupyter                       1.0.0
jupyter-bokeh                 3.0.5
jupyter-console               6.6.3
jupyter-dash                  0.4.2
jupyter-resource-usage        0.7.2
jupyter-server                1.24.0
jupyter_client                8.2.0
jupyter_core                  5.3.0
jupyterlab                    3.4.4
jupyterlab-pygments           0.2.2
jupyterlab-widgets            3.0.7
jupyterlab_server             2.22.1
keract                        4.5.1
keras                         2.11.0
keras-nlp                     0.4.1
keras-rl                      0.4.2
keras-tuner                   1.3.5
kiwisolver                    1.4.4
kmapper                       2.0.1
korean-lunar-calendar         0.3.1
kt-legacy                     1.0.5
langcodes                     3.3.0
lark                          1.1.5
lazy_loader                   0.2
lazypredict                   0.2.12
libclang                      16.0.0
lightgbm                      3.3.5
lime                          0.2.0.1
line-profiler                 4.0.2
linear-operator               0.3.0
littleutils                   0.2.2
livelossplot                  0.5.5
llvmlite                      0.39.1
locket                        1.0.0
logical-unification           0.4.5
LunarCalendar                 0.0.9
lxml                          4.9.2
lz4                           4.3.2
Mako                          1.2.4
Markdown                      3.4.3
MarkupSafe                    2.1.2
marshmallow                   3.19.0
matplotlib                    3.7.0
matplotlib-inline             0.1.6
miniful                       0.0.6
miniKanren                    1.0.3
minorminer                    0.2.9
mistune                       2.0.5
```

```
mljar-supervised              0.11.5
mlxtend                       0.21.0
mmh3                          2.5.1
modin                         0.18.1
mpi4py                        3.1.4
mplfinance                    0.12.9b7
mpmath                        1.2.1
msgpack                       1.0.4
mthree                        2.4.0
multidict                     6.0.4
multipledispatch              0.6.0
multiprocess                  0.70.14
multitasking                  0.0.11
murmurhash                    1.0.9
nbclassic                     0.5.5
nbclient                      0.7.3
nbconvert                     7.3.1
nbeats-keras                  1.8.0
nbeats-pytorch                1.8.0
nbformat                      5.8.0
nest-asyncio                  1.5.6
networkx                      2.8.8
neural-tangents               0.6.2
neuralprophet                 0.5.0
nfoursid                      1.0.1
nltk                          3.8.1
nose                          1.3.7
notebook                      6.5.4
notebook_shim                 0.2.2
ntlm-auth                     1.5.0
numba                         0.56.4
numerapi                      2.13.2
numexpr                       2.8.4
numpy                         1.23.5
numpydoc                      1.5.0
nvidia-cublas-cu11            11.10.3.66
nvidia-cuda-nvrtc-cu11        11.7.99
nvidia-cuda-runtime-cu11      11.7.99
nvidia-cudnn-cu11             8.5.0.96
nvidia-ml-py                  11.525.112
oauthlib                      3.2.2
openai                        0.26.5
opencensus                    0.11.2
opencensus-context            0.1.3
opencv-python                 4.7.0.72
openpyxl                      3.1.1
opt-einsum                    3.3.0
optuna                        3.1.0
orjson                        3.8.10
ortools                       9.4.1874
osqp                          0.6.2.post9
outdated                      0.2.2
packaging                     23.1
pandas                        1.5.3
pandas-datareader             0.10.0
pandas-flavor                 0.5.0
pandas-market-calendars       4.1.4
pandas-ta                     0.3.14b0
pandocfilters                 1.5.0
panel                         0.14.3
param                         1.13.0
parso                         0.8.3
partd                         1.4.0
pastel                        0.2.1
pathos                        0.3.0
pathy                         0.10.1
patsy                         0.5.3
pbr                           5.11.1
penaltymodel                  1.0.2
PennyLane                     0.29.0
PennyLane-Lightning           0.29.0
PennyLane-qiskit              0.29.0
persim                        0.3.1
pexpect                       4.8.0
pickleshare                   0.7.5
Pillow                        9.5.0
pingouin                      0.5.3
pip                           22.3.1
pkgutil_resolve_name          1.3.10
platformdirs                  3.2.0
```

```
plotly                          5.13.1
plotly-resampler                0.8.3.2
plucky                          0.4.3
pluggy                          1.0.0
ply                             3.11
pmdarima                        2.0.2
polars                          0.16.9
pox                             0.3.2
ppft                            1.7.6.6
pprofile                        2.1.0
ppscore                         1.2.0
preshed                         3.0.8
prometheus-client               0.16.0
prompt-toolkit                  3.0.38
property-cached                 1.6.4
prophet                         1.1.2
protobuf                        3.19.6
psutil                          5.9.5
psycopg2-binary                 2.9.6
ptvsd                           4.3.2
ptyprocess                      0.7.0
PuLP                            2.7.0
pure-eval                       0.2.2
py-cpuinfo                      9.0.0
py-heat                         0.0.6
py-heat-magic                   0.0.2
py-lets-be-rational             1.0.1
py-spy                          0.3.14
py-vollib                       1.0.1
py4j                            0.10.9.7
pyaml                           21.10.1
pyarrow                         11.0.0
pyasn1                          0.4.8
pyasn1-modules                  0.2.8
pybind11                        2.10.4
pycares                         4.3.0
pycosat                         0.6.4
pycparser                       2.21
pyct                            0.5.0
pydantic                        1.10.7
pyDeprecate                     0.3.2
pydevd-pycharm                  201.8538.36
pydmd                           0.4.0.post2301
pyerfa                          2.0.0.3
pyfolio                         0.9.2
pyfolio-reloaded                0.9.5
pyFUME                          0.2.25
Pygments                        2.15.1
pykalman                        0.9.5
pylev                           1.4.0
pyluach                         2.2.0
pymannkendall                   1.4.3
pymc                            5.0.2
pymdptoolbox                    4.0b3
PyMeeus                         0.5.12
PyMySQL                         1.0.3
pynndescent                     0.5.9
pyod                            1.0.9
Pyomo                           6.5.0
pyOpenSSL                       22.0.0
pyparsing                       3.0.9
pyportfolioopt                  1.5.4
pyqubo                          1.3.1
pyrb                            1.0.1
pyro-api                        0.1.2
pyro-ppl                        1.8.4
pyrsistent                      0.19.3
pysimdjson                      5.0.2
PySocks                         1.7.1
pystan                          3.6.0
pytensor                        2.9.1
pytest                          7.3.1
python-dateutil                 2.8.2
python-statemachine             1.0.3
pytorch-ignite                  0.4.11
pytorch-lightning               1.7.4
pytz                            2023.3
pyvinecopulib                   0.6.2
pyviz-comms                     2.2.1
PyWavelets                      1.4.1
```

```
PyYAML                          6.0
pyzmq                           25.0.2
qdldl                           0.1.7
qiskit                          0.41.1
qiskit-aer                      0.11.2
qiskit-ibmq-provider            0.20.1
qiskit-terra                    0.23.2
qpd                             0.4.1
qtconsole                       5.4.2
QtPy                            2.3.1
quadprog                        0.1.11
quantecon                       0.6.0
QuantLib                        1.29
QuantStats                      0.0.59
rauth                           0.7.3
ray                             2.3.0
rectangle-packer                2.0.1
regex                           2023.3.23
requests                        2.28.1
requests-ntlm                   1.1.0
requests-oauthlib               1.3.1
retrying                        1.3.4
retworkx                        0.12.1
rich                            12.4.4
ripser                          0.6.4
Riskfolio-Lib                   4.0.3
riskparityportfolio             0.4
river                           0.14.0
rsa                             4.9
ruamel.yaml                     0.17.21
ruamel.yaml.clib                0.2.6
ruptures                        1.1.7
rustworkx                       0.12.1
SALib                           1.4.7
scikeras                        0.10.0
scikit-image                    0.19.3
scikit-learn                    1.2.1
scikit-learn-extra              0.2.0
scikit-multiflow                0.5.3
scikit-optimize                 0.9.0
scikit-plot                     0.3.7
scikit-tda                      1.0.0
scipy                           1.10.1
scs                             3.2.3
sdeint                          0.3.0
seaborn                         0.12.2
semantic-version                2.10.0
Send2Trash                      1.8.0
setuptools                      64.0.2
setuptools-scm                  7.1.0
shap                            0.41.0
simpful                         2.10.0
simplejson                      3.19.1
simpy                           4.0.1
six                             1.16.0
sklearn-json                    0.1.0
skope-rules                     1.0.1
sktime                          0.16.1
slicer                          0.0.7
smart-open                      6.3.0
sniffio                         1.3.0
snowballstemmer                 2.2.0
sortedcontainers                2.4.0
soupsieve                       2.4.1
spacy                           3.5.0
spacy-legacy                    3.0.12
spacy-loggers                   1.0.4
Sphinx                          6.1.3
sphinxcontrib-applehelp         1.0.4
sphinxcontrib-devhelp           1.0.2
sphinxcontrib-htmlhelp          2.0.1
sphinxcontrib-jsmath            1.0.1
sphinxcontrib-qthelp            1.0.3
sphinxcontrib-serializinghtml 1.1.5
SQLAlchemy                      1.4.47
sqlglot                         11.5.5
srsly                           2.4.6
ssm                             0.0.1
stable-baselines3               1.7.0
stack-data                      0.6.2
```

```
statsforecast                 1.5.0
statsmodels                   0.13.5
stellargraph                  1.2.1
stevedore                     5.0.0
stochastic                    0.7.0
stockstats                    0.5.2
stumpy                        1.11.1
symengine                     0.10.0
sympy                         1.11.1
ta                            0.10.2
TA-Lib                        0.4.18
tables                        3.8.0
tabulate                      0.8.10
tadasets                      0.0.4
tbats                         1.1.3
tblib                         1.7.0
tenacity                      8.2.2
tensorboard                   2.11.2
tensorboard-data-server       0.6.1
tensorboard-plugin-wit        1.8.1
tensorboardX                  2.6
tensorflow                    2.11.0
tensorflow-addons             0.19.0
tensorflow-decision-forests   1.2.0
tensorflow-estimator          2.11.0
tensorflow-hub                0.13.0
tensorflow-io-gcs-filesystem  0.32.0
tensorflow-probability        0.19.0
tensorflow-text               2.11.0
tensorly                      0.8.0
tensortrade                   1.0.3
termcolor                     2.2.0
terminado                     0.17.1
tf2jax                        0.3.3
thinc                         8.1.9
threadpoolctl                 3.1.0
thriftpy2                     0.4.16
thundergbm                    0.3.17
tick                          0.7.0.1
tifffile                      2023.4.12
tinycss2                      1.2.1
toml                          0.10.2
tomli                         2.0.1
toolz                         0.12.0
torch                         1.13.1
torch-cluster                 1.6.0
torch-geometric               2.2.0
torch-scatter                 2.1.0
torch-sparse                  0.6.16
torch-spline-conv             1.2.1
torchmetrics                  0.9.3
torchvision                   0.14.1
tornado                       6.3
tqdm                          4.64.1
trace-updater                 0.0.9.1
trading-calendars             2.1.1
traitlets                     5.9.0
treeinterpreter               0.2.3
triad                         0.8.6
tsfresh                       0.20.0
tslearn                       0.5.3.2
tweepy                        4.10.0
typeguard                     3.0.2
typer                         0.3.2
typing_extensions             4.5.0
umap-learn                    0.5.3
urllib3                       1.26.14
virtualenv                    20.21.0
wasabi                        1.1.1
wcwidth                       0.2.6
webargs                       8.2.0
webencodings                  0.5.1
websocket-client              1.5.1
websockets                    10.4
Werkzeug                      2.2.3
wheel                         0.37.1
widgetsnbextension            4.0.7
wordcloud                     1.8.2.2
wrapt                         1.14.1
wrds                          3.1.6
```

```
wurlitzer                     3.0.3
xarray                        2023.1.0
xarray-einstats               0.5.1
xgboost                       1.7.4
xlrd                          2.0.1
XlsxWriter                    3.1.0
yarl                          1.8.2
yellowbrick                   1.5
yfinance                      0.2.18
zict                          3.0.0
zipp                          3.15.0
zope.event                    4.6
zope.interface                6.0
zstandard                     0.18.0
```

Pomegranate Environment Libraries

The Pomegranate environment supports the following libraries:

PY

```
# Name                        Version
absl-py                       1.4.0
adagio                        0.2.4
aesara                        2.8.12
aiodns                        3.0.0
aiohttp                       3.8.4
aiohttp-cors                  0.7.0
aiosignal                     1.3.1
alabaster                     0.7.13
ale-py                        0.7.4
alembic                       1.10.3
alpaca-trade-api              0.26
alphalens-reloaded            0.4.3
altair                        4.2.2
ansi2html                     1.8.0
antlr4-python3-runtime        4.11.1
anyio                         3.6.2
appdirs                       1.4.4
arch                          5.3.1
argon2-cffi                   21.3.0
argon2-cffi-bindings          21.2.0
arviz                         0.15.1
astropy                       5.2.1
asttokens                     2.2.1
astunparse                    1.6.3
async-timeout                 4.0.2
asyncio-nats-client           0.11.5
attrs                         23.1.0
autograd                      1.5
autokeras                     1.1.0
autoray                       0.6.3
AutoROM                       0.4.2
AutoROM.accept-rom-license    0.6.1
ax-platform                   0.3.0
Babel                         2.12.1
backcall                      0.2.0
bayesian-optimization         1.4.2
beautifulsoup4                4.11.2
bleach                        6.0.0
blessed                       1.20.0
blis                          0.7.9
blosc2                        2.0.0
bokeh                         2.4.3
boltons                       23.0.0
botorch                       0.8.2
Bottleneck                    1.3.7
brotlipy                      0.7.0
cachetools                    5.3.0
captum                        0.6.0
catalogue                     2.0.8
catboost                      1.1.1
category-encoders             2.6.0
ccxt                          3.0.72
certifi                       2022.12.7
cffi                          1.15.1
chardet                       5.1.0
```

```
chardet                   5.1.0
charset-normalizer        2.0.4
check-shapes              1.0.0
click                     7.1.2
clikit                    0.6.2
cloudpickle               2.2.1
cmaes                     0.9.1
cmdstanpy                 1.1.0
colorama                  0.4.6
colorcet                  3.0.1
colorful                  0.5.5
colorlog                  6.7.0
colorlover                0.3.0
colour                    0.1.5
comm                      0.1.3
commonmark                0.9.1
conda                     23.3.1
conda-content-trust       0.1.3
conda-package-handling    2.0.2
conda_package_streaming   0.7.0
confection                0.0.4
cons                      0.4.5
contourpy                 1.0.7
convertdate               2.4.0
copulae                   0.7.7
copulas                   0.8.0
cramjam                   2.6.2
crashtest                 0.3.1
creme                     0.6.1
cryptography              38.0.4
cufflinks                 0.17.3
cvxopt                    1.3.0
cvxpy                     1.3.0
cycler                    0.11.0
cymem                     2.0.7
Cython                    0.29.33
darts                     0.23.1
dash                      2.9.3
dash-core-components      2.0.0
dash-cytoscape            0.3.0
dash-html-components      2.0.0
dash-table                5.0.0
dask                      2023.4.0
deap                      1.3.3
debugpy                   1.5.1
decorator                 5.1.1
defusedxml                0.7.1
Deprecated                1.2.13
dgl                       1.0.1
dill                      0.3.6
dimod                     0.12.3
diskcache                 5.6.1
distlib                   0.3.6
distributed               2021.4.1
dm-tree                   0.1.8
docutils                  0.19
DoubleML                  0.5.2
dtreeviz                  2.2.1
dtw-python                1.3.0
dwave-cloud-client        0.10.3
dwave-drivers             0.4.4
dwave-greedy              0.3.0
dwave-hybrid              0.6.9
dwave-inspector           0.3.0
dwave-inspectorapp        0.3.0
dwave-neal                0.6.0
dwave-networkx            0.8.12
dwave-ocean-sdk           6.1.1
dwave-preprocessing       0.5.3
dwave-samplers            1.0.0
dwave-system              1.18.0
dwave-tabu                0.5.0
dwavebinarycsp            0.2.0
dx                        0.1.22
ecos                      2.0.12
EMD-signal                1.4.0
empyrical                 0.5.5
empyrical-reloaded        0.5.9
en-core-web-md            3.5.0
en-core-web-sm            3.5.0
entrypoints               0.4
```

```
entrypoints                   0.4
ephem                         4.1.4
et-xmlfile                    1.1.0
etuples                       0.3.8
exceptiongroup                1.1.1
exchange-calendars            4.2.6
executing                     1.2.0
ezyrb                         1.3.0.post2304
fastai                        2.7.11
fastai2                       0.0.30
fastcore                      1.5.29
fastdownload                  0.0.7
fasteners                     0.18
fastjsonschema                2.16.3
fastparquet                   2023.2.0
fastprogress                  1.0.3
fasttext                      0.9.2
featuretools                  0.23.1
filelock                      3.12.0
findiff                       0.9.2
finrl                         0.3.1
FixedEffectModel              0.0.5
Flask                         2.0.3
flatbuffers                   23.3.3
fonttools                     4.39.3
fracdiff                      0.9.0
frozendict                    2.3.7
frozenlist                    1.3.3
fs                            2.4.16
fsspec                        2023.4.0
fst-pso                       1.8.1
fugue                         0.8.3
fugue-sql-antlr               0.1.6
future                        0.18.3
fuzzy-c-means                 1.6.3
FuzzyTM                       2.0.5
gast                          0.4.0
gensim                        4.3.0
gevent                        22.10.2
gluonts                       0.12.3
google-api-core               2.11.0
google-auth                   2.17.3
google-auth-oauthlib          0.4.6
google-pasta                  0.2.0
googleapis-common-protos      1.59.0
gpflow                        2.7.0
gplearn                       0.4.2
gpustat                       1.1
GPUtil                        1.4.0
gpytorch                      1.9.1
graphviz                      0.8.4
greenlet                      2.0.2
grpcio                        1.54.0
gym                           0.21.0
Gymnasium                     0.26.3
gymnasium-notices             0.0.1
h2o                           3.40.0.1
h5netcdf                      1.1.0
h5py                          3.8.0
hijri-converter               2.2.4
hmmlearn                      0.2.8
holidays                      0.23
holoviews                     1.15.4
homebase                      1.0.1
hopcroftkarp                  1.2.5
html5lib                      1.1
httpstan                      4.9.1
hurst                         0.0.5
hvplot                        0.8.2
hyperopt                      0.2.7
idna                          3.4
iisignature                   0.24
ijson                         3.2.0.post0
imageio                       2.27.0
imagesize                     1.4.1
imbalanced-learn              0.10.1
importlib-metadata            4.13.0
importlib-resources           5.12.0
iniconfig                     2.0.0
injector                      0.20.1
interpret                     0.3.0
```

```
interpret                 0.3.0
interpret-core            0.3.2
ipykernel                 6.22.0
ipython                   8.12.0
ipython-genutils          0.2.0
ipywidgets                8.0.4
itsdangerous              2.1.2
jax                       0.4.4
jaxlib                    0.4.4
jedi                      0.18.2
Jinja2                    3.1.2
joblib                    1.2.0
jqdatasdk                 1.8.11
json5                     0.9.11
jsonpatch                 1.32
jsonpointer               2.0
jsonschema                4.17.3
jupyter                   1.0.0
jupyter-bokeh             3.0.5
jupyter-console           6.6.3
jupyter-dash              0.4.2
jupyter-resource-usage    0.7.2
jupyter-server            1.24.0
jupyter_client            8.2.0
jupyter_core              5.3.0
jupyterlab                3.4.4
jupyterlab-pygments       0.2.2
jupyterlab-widgets        3.0.7
jupyterlab_server         2.22.1
keract                    4.5.1
keras                     2.11.0
keras-nlp                 0.4.1
keras-rl                  0.4.2
keras-tuner               1.3.5
kiwisolver                1.4.4
kmapper                   2.0.1
korean-lunar-calendar     0.3.1
kt-legacy                 1.0.5
langcodes                 3.3.0
lark                      1.1.5
lazy_loader               0.2
lazypredict               0.2.12
libclang                  16.0.0
lightgbm                  3.3.5
lime                      0.2.0.1
line-profiler             4.0.2
linear-operator           0.3.0
littleutils               0.2.2
livelossplot              0.5.5
llvmlite                  0.38.1
locket                    1.0.0
logical-unification       0.4.5
LunarCalendar             0.0.9
lxml                      4.9.2
lz4                       4.3.2
Mako                      1.2.4
Markdown                  3.4.3
MarkupSafe                2.1.2
marshmallow               3.19.0
matplotlib                3.7.0
matplotlib-inline         0.1.6
miniful                   0.0.6
miniKanren                1.0.3
minorminer                0.2.9
mistune                   2.0.5
mljar-supervised          0.11.5
mlxtend                   0.21.0
mmh3                      2.5.1
modin                     0.18.1
mpi4py                    3.1.4
mplfinance                0.12.9b7
mpmath                    1.2.1
msgpack                   1.0.4
mthree                    2.4.0
multidict                 6.0.4
multipledispatch          0.6.0
multiprocess              0.70.14
multitasking              0.0.11
murmurhash                1.0.9
mxnet                     1.9.1
nbclassic                 0.5.5
```

```
nbclassic                     0.5.5
nbclient                      0.7.3
nbconvert                     7.3.1
nbeats-keras                  1.8.0
nbeats-pytorch                1.8.0
nbformat                      5.8.0
nest-asyncio                  1.5.6
networkx                      2.8.8
neural-tangents               0.6.2
neuralprophet                 0.5.0
nfoursid                      1.0.1
nltk                          3.8.1
nose                          1.3.7
notebook                      6.5.4
notebook_shim                 0.2.2
ntlm-auth                     1.5.0
numba                         0.55.1
numerapi                      2.13.2
numexpr                       2.8.4
numpy                         1.21.5
numpydoc                      1.5.0
nvidia-cublas-cu11            11.10.3.66
nvidia-cuda-nvrtc-cu11        11.7.99
nvidia-cuda-runtime-cu11      11.7.99
nvidia-cudnn-cu11             8.5.0.96
nvidia-ml-py                  11.525.112
oauthlib                      3.2.2
openai                        0.26.5
opencensus                    0.11.2
opencensus-context            0.1.3
opencv-python                 4.7.0.72
openpyxl                      3.1.1
opt-einsum                    3.3.0
optuna                        3.1.0
orjson                        3.8.10
ortools                       9.4.1874
osqp                          0.6.2.post9
outdated                      0.2.2
packaging                     23.1
pandas                        1.5.3
pandas-datareader             0.10.0
pandas-flavor                 0.5.0
pandas-market-calendars       4.1.4
pandas-ta                     0.3.14b0
pandocfilters                 1.5.0
panel                         0.14.3
param                         1.13.0
parso                         0.8.3
partd                         1.4.0
pastel                        0.2.1
pathos                        0.3.0
pathy                         0.10.1
patsy                         0.5.3
pbr                           5.11.1
penaltymodel                  1.0.2
PennyLane                     0.29.0
PennyLane-Lightning           0.29.0
PennyLane-qiskit              0.29.0
persim                        0.3.1
pexpect                       4.8.0
pickleshare                   0.7.5
Pillow                        9.5.0
pingouin                      0.5.3
pip                           22.0.4
pkgutil_resolve_name          1.3.10
platformdirs                  3.2.0
plotly                        5.13.1
plotly-resampler              0.8.3.2
plucky                        0.4.3
pluggy                        1.0.0
ply                           3.11
pmdarima                      2.0.2
polars                        0.16.9
pomegranate                   0.14.8
pox                           0.3.2
ppft                          1.7.6.6
pprofile                      2.1.0
ppscore                       1.2.0
preshed                       3.0.8
prometheus-client             0.16.0
prompt-toolkit                3.0.38
```

```
prompt-toolkit                3.0.38
property-cached               1.6.4
prophet                       1.1.2
protobuf                      3.19.6
psutil                        5.9.5
psycopg2-binary               2.9.6
ptvsd                         4.3.2
ptyprocess                    0.7.0
PuLP                          2.7.0
pure-eval                     0.2.2
py-cpuinfo                    9.0.0
py-heat                       0.0.6
py-heat-magic                 0.0.2
py-lets-be-rational           1.0.1
py-spy                        0.3.14
py-vollib                     1.0.1
py4j                          0.10.9.7
pyaml                         21.10.1
pyarrow                       11.0.0
pyasn1                        0.4.8
pyasn1-modules                0.2.8
pybind11                      2.10.4
pycares                       4.3.0
pycosat                       0.6.4
pycparser                     2.21
pyct                          0.5.0
pydantic                      1.10.7
pyDeprecate                   0.3.2
pydevd-pycharm                201.8538.36
pydmd                         0.4.0.post2301
pyerfa                        2.0.0.3
pyfolio                       0.9.2
pyfolio-reloaded              0.9.5
pyFUME                        0.2.25
Pygments                      2.15.1
pykalman                      0.9.5
pylev                         1.4.0
pyluach                       2.2.0
pymannkendall                 1.4.3
pymc                          5.0.2
pymdptoolbox                  4.0b3
PyMeeus                       0.5.12
PyMySQL                       1.0.3
pynndescent                   0.5.9
pyod                          1.0.9
Pyomo                         6.5.0
pyOpenSSL                     22.0.0
pyparsing                     3.0.9
pyportfolioopt                1.5.4
pyqubo                        1.3.1
pyrb                          1.0.1
pyro-api                      0.1.2
pyro-ppl                      1.8.4
pyrsistent                    0.19.3
pysimdjson                    5.0.2
PySocks                       1.7.1
pystan                        3.6.0
pytensor                      2.9.1
pytest                        7.3.1
python-dateutil               2.8.2
python-statemachine           1.0.3
pytorch-ignite                0.4.11
pytorch-lightning             1.7.4
pytz                          2023.3
pyvinecopulib                 0.6.2
pyviz-comms                   2.2.1
PyWavelets                    1.4.1
PyYAML                        6.0
pyzmq                         25.0.2
qdldl                         0.1.7
qiskit                        0.41.1
qiskit-aer                    0.11.2
qiskit-ibmq-provider          0.20.1
qiskit-terra                  0.23.2
qpd                           0.4.1
qtconsole                     5.4.2
QtPy                          2.3.1
quadprog                      0.1.11
quantecon                     0.6.0
QuantLib                      1.29
QuantStats                    0.0.59
```

```
quantstats                      0.0.59
rauth                           0.7.3
ray                             2.3.0
rectangle-packer                2.0.1
regex                           2023.3.23
requests                        2.28.1
requests-ntlm                   1.1.0
requests-oauthlib               1.3.1
retrying                        1.3.4
retworkx                        0.12.1
rich                            12.4.4
ripser                          0.6.4
Riskfolio-Lib                   4.0.3
riskparityportfolio             0.4
river                           0.14.0
rsa                             4.9
ruamel.yaml                     0.17.21
ruamel.yaml.clib                0.2.6
ruptures                        1.1.7
rustworkx                       0.12.1
SALib                           1.4.7
scikeras                        0.10.0
scikit-image                    0.19.3
scikit-learn                    1.2.1
scikit-learn-extra              0.2.0
scikit-multiflow                0.5.3
scikit-optimize                 0.9.0
scikit-plot                     0.3.7
scikit-tda                      1.0.0
scipy                           1.8.0
scs                             3.2.3
sdeint                          0.3.0
seaborn                         0.12.2
semantic-version                2.10.0
Send2Trash                      1.8.0
setuptools                      56.0.0
setuptools-scm                  7.1.0
shap                            0.41.0
simpful                         2.10.0
simplejson                      3.19.1
simpy                           4.0.1
six                             1.16.0
sklearn-json                    0.1.0
skope-rules                     1.0.1
sktime                          0.16.1
slicer                          0.0.7
smart-open                      6.3.0
sniffio                         1.3.0
snowballstemmer                 2.2.0
sortedcontainers                2.4.0
soupsieve                       2.4.1
spacy                           3.5.0
spacy-legacy                    3.0.12
spacy-loggers                   1.0.4
Sphinx                          6.1.3
sphinxcontrib-applehelp         1.0.4
sphinxcontrib-devhelp           1.0.2
sphinxcontrib-htmlhelp          2.0.1
sphinxcontrib-jsmath            1.0.1
sphinxcontrib-qthelp            1.0.3
sphinxcontrib-serializinghtml 1.1.5
SQLAlchemy                      1.4.47
sqlglot                         11.5.5
srsly                           2.4.6
ssm                             0.0.1
stable-baselines3               1.7.0
stack-data                      0.6.2
statsforecast                   1.5.0
statsmodels                     0.13.5
stellargraph                    1.2.1
stevedore                       5.0.0
stochastic                      0.7.0
stockstats                      0.5.2
stumpy                          1.11.1
symengine                       0.10.0
sympy                           1.11.1
ta                              0.10.2
TA-Lib                          0.4.18
tables                          3.8.0
tabulate                        0.8.10
tadasets                        0.0.4
```

```
tadasets                  0.0.4
tbats                     1.1.3
tblib                     1.7.0
tenacity                  8.2.2
tensorboard               2.11.2
tensorboard-data-server   0.6.1
tensorboard-plugin-wit    1.8.1
tensorboardX              2.6
tensorflow                2.11.0
tensorflow-addons         0.19.0
tensorflow-decision-forests 1.2.0
tensorflow-estimator      2.11.0
tensorflow-hub            0.13.0
tensorflow-io-gcs-filesystem 0.32.0
tensorflow-probability    0.19.0
tensorflow-text           2.11.0
tensorly                  0.8.0
tensortrade               1.0.3
termcolor                 2.2.0
terminado                 0.17.1
tf2jax                    0.3.3
thinc                     8.1.9
threadpoolctl             3.1.0
thriftpy2                 0.4.16
thundergbm                0.3.17
tick                      0.7.0.1
tifffile                  2023.4.12
tigramite                 5.1.0.3
tinycss2                  1.2.1
toml                      0.10.2
tomli                     2.0.1
toolz                     0.12.0
torch                     1.13.1
torch-cluster             1.6.0
torch-geometric           2.2.0
torch-scatter             2.1.0
torch-sparse              0.6.16
torch-spline-conv         1.2.1
torchmetrics              0.9.3
torchvision               0.14.1
tornado                   6.3
tqdm                      4.64.1
trace-updater             0.0.9.1
trading-calendars         2.1.1
traitlets                 5.9.0
treeinterpreter           0.2.3
triad                     0.8.6
tsfresh                   0.20.0
tslearn                   0.5.3.2
tweepy                    4.10.0
typeguard                 3.0.2
typer                     0.3.2
typing_extensions         4.5.0
umap-learn                0.5.3
urllib3                   1.26.14
virtualenv                20.21.0
wasabi                    1.1.1
wcwidth                   0.2.6
webargs                   8.2.0
webencodings              0.5.1
websocket-client          1.5.1
websockets                10.4
Werkzeug                  2.2.3
wheel                     0.37.1
widgetsnbextension        4.0.7
wordcloud                 1.8.2.2
wrapt                     1.14.1
wrds                      3.1.6
wurlitzer                 3.0.3
xarray                    2023.1.0
xarray-einstats           0.5.1
xgboost                   1.7.4
xlrd                      2.0.1
XlsxWriter                3.1.0
yarl                      1.8.2
yellowbrick               1.5
yfinance                  0.2.18
zict                      3.0.0
zipp                      3.15.0
zope.event                4.6
zope.interface            6.0
```

```
zope.interface                 6.0
zstandard                      0.18.0
```

Hudson & Thames Environment

The Hudson & Thames environment supports the following libraries:

```PY
# Name                         Version
absl-py                        1.3.0
analytics-python               1.4.0
ansi2html                      1.8.0
anyio                          3.6.2
appdirs                        1.4.4
arch                           4.16.1
argon2-cffi                    21.3.0
argon2-cffi-bindings           21.2.0
asttokens                      2.2.1
astunparse                     1.6.3
attrs                          22.2.0
Babel                          2.11.0
backcall                       0.2.0
backoff                        1.10.0
beautifulsoup4                 4.11.1
bleach                         5.0.1
Brotli                         1.0.9
bs4                            0.0.1
cachetools                     5.2.0
certifi                        2022.12.7
cffi                           1.15.1
charset-normalizer             2.1.1
click                          8.1.3
cloudpickle                    2.2.0
clr-loader                     0.1.6
comm                           0.1.2
cramjam                        2.6.2
cssselect                      1.2.0
cvxpy                          1.2.0
cycler                         0.11.0
Cython                         0.29.28
dash                           1.21.0
dash-bootstrap-components      0.13.1
dash-core-components           1.17.1
dash-cytoscape                 0.2.0
dash-html-components           1.1.4
dash-table                     4.12.0
dask                           2022.12.1
debugpy                        1.6.4
decorator                      4.4.2
defusedxml                     0.7.1
distributed                    2022.12.1
ecos                           2.0.12
entrypoints                    0.4
executing                      1.2.0
fake-useragent                 1.1.1
fastjsonschema                 2.16.2
fastparquet                    0.8.1
Flask                          2.1.3
Flask-Compress                 1.13
flatbuffers                    22.12.6
fsspec                         2022.11.0
future                         0.18.2
gast                           0.5.3
getmac                         0.8.3
google-auth                    2.15.0
google-auth-oauthlib           0.4.6
google-pasta                   0.2.0
grpcio                         1.51.1
h5py                           3.7.0
HeapDict                       1.0.1
ht-auth                        0.1.0
idna                           3.4
importlib-metadata             5.2.0
importlib-resources            5.10.1
ipykernel                      6.19.4
ipython                        8.7.0
ipython-genutils               0.2.0
```

```
ipython-genutils            0.2.0
itsdangerous                2.1.2
jedi                        0.18.2
Jinja2                      3.1.2
joblib                      1.2.0
json5                       0.9.10
jsonschema                  4.17.3
jupyter_client              7.4.8
jupyter_core                5.1.1
jupyter-dash                0.4.2
jupyter-server              1.23.4
jupyterlab                  3.4.4
jupyterlab-pygments         0.2.2
jupyterlab_server           2.17.0
keras                       2.8.0
Keras-Preprocessing         1.1.2
kiwisolver                  1.4.4
libclang                    14.0.6
llvmlite                    0.39.1
locket                      1.0.0
lxml                        4.6.2
Markdown                    3.4.1
MarkupSafe                  2.1.1
matplotlib                  3.4.3
matplotlib-inline           0.1.6
matrixprofile               1.1.10
mistune                     2.0.4
mlfinlab                    1.6.0
monotonic                   1.6
mpmath                      1.2.1
msgpack                     1.0.4
multitasking                0.0.11
nbclassic                   0.4.8
nbclient                    0.7.2
nbconvert                   7.2.7
nbformat                    5.7.1
nest-asyncio                1.5.6
networkx                    2.5.1
notebook                    6.5.2
notebook_shim               0.2.2
numba                       0.56.4
numpy                       1.20.1
oauthlib                    3.2.2
opt-einsum                  3.3.0
osqp                        0.6.2.post8
packaging                   22.0
pandas                      1.1.5
pandocfilters               1.5.0
parse                       1.19.0
parso                       0.8.3
partd                       1.3.0
patsy                       0.5.3
pexpect                     4.8.0
pickleshare                 0.7.5
Pillow                      9.3.0
pip                         22.0.4
pkgutil_resolve_name        1.3.10
platformdirs                2.6.1
plotly                      5.11.0
pmdarima                    1.8.5
POT                         0.8.2
prometheus-client           0.15.0
prompt-toolkit              3.0.36
property-cached             1.6.4
protobuf                    3.11.2
psutil                      5.9.4
ptyprocess                  0.7.0
pure-eval                   0.2.2
pyasn1                      0.4.8
pyasn1-modules              0.2.8
pybind11                    2.10.1
pycparser                   2.21
pyee                        8.2.2
Pygments                    2.13.0
pykalman                    0.9.5
pyparsing                   3.0.9
pyppeteer                   1.0.2
pyquery                     2.0.0
pyrsistent                  0.19.2
python-dateutil             2.8.2
pytz                        2022.7
```

```
pytz                            2022.7
pyvinecopulib                   0.5.5
PyYAML                          6.0
pyzmq                           24.0.1
qdldl                           0.1.5.post2
requests                        2.28.1
requests-html                   0.10.0
requests-oauthlib               1.3.1
retrying                        1.3.4
rsa                             4.9
scikit-learn                    0.24.2
scipy                           1.9.3
scs                             3.2.0
seaborn                         0.11.1
Send2Trash                      1.8.0
setuptools                      56.0.0
setuptools-scm                  7.1.0
six                             1.16.0
sniffio                         1.3.0
sortedcontainers                2.4.0
soupsieve                       2.3.2.post1
stack-data                      0.6.2
statsmodels                     0.13.5
stumpy                          1.11.1
TA-Lib                          0.4.18
tblib                           1.7.0
tenacity                        8.1.0
tensorboard                     2.8.0
tensorboard-data-server         0.6.1
tensorboard-plugin-wit          1.8.1
tensorflow                      2.8.0
tensorflow-io-gcs-filesystem 0.29.0
termcolor                       2.1.1
terminado                       0.17.1
tf-estimator-nightly            2.8.0.dev2021122109
threadpoolctl                   3.1.0
tinycss2                        1.2.1
tomli                           2.0.1
toolz                           0.12.0
tornado                         6.2
tqdm                            4.64.1
traitlets                       5.8.0
tsfresh                         0.19.0
typing_extensions               4.4.0
urllib3                         1.26.13
w3lib                           2.1.1
wcwidth                         0.2.5
webencodings                    0.5.1
websocket-client                1.4.2
websockets                      10.4
Werkzeug                        2.0.0
wheel                           0.38.4
wrapt                           1.14.1
yahoo-fin                       0.8.6
yfinance                        0.1.63
zict                            2.2.0
zipp                            3.11.0
```

Add New Libraries

To request a new library, contact us . We will add the library to the queue for review and deployment. Since the libraries run on our servers, we need to ensure they are secure and won't cause harm. The process of adding new libraries takes 2-4 weeks to complete. View the list of libraries currently under review on the Issues list of the Lean GitHub repository .

Project Libraries

Project libraries are QuantConnect projects you can merge into your project to avoid duplicating code files. If you have tools that you use across several projects, create a library.

Key Concepts

Glossary

Introduction

This page defines terms in QuantConnect products and documentation.

alpha

The quantity of an algorithm's returns that aren't explained by its underlying benchmark.

annual standard deviation

A staticial measure that describes the dispersion of annual returns relative to the mean annual return. It's the square root of the annual variance.

annual variance

A staticial measure that describes the dispersion of annual returns relative to the mean annual return.

average loss

The average rate of return for unprofitable trades.

average win

The average rate of return for profitable trades.

beta

The scale and direction of an algorithm's returns relative to movements in the underlying benchmark.

capacity

The maximum amount of money an algorithm can trade before its performance degrades from market impact.

compounding annual return

The annual percentage return that would be required to grow a portfolio from its starting value to its ending value.

day trade

Buying and selling the same asset within one trading day.

drawdown

The largest peak to trough decline in an algorithm's equity curve.

equity

The total portfolio value if all of the holdings were sold at current market rates.

expectancy

The expected return per trade.

holdings

The absolute sum of the items in the portfolio.

information ratio

The amount of excess return from the risk-free rate per unit of systematic risk.

look-ahead bias

The practice of making decisions using information that would not be available until some time in the future.

loss rate

The proportion of trades that were not profitable.

lowest capacity asset

The asset an algorithm traded that has the lowest capacity.

net profit

(Percent) The rate of return across the entire trading period.

(Dollar-value) The dollar-value return across the entire trading period.

pattern day trader

A trader who executes four or more day trades in the US Equity market within five business days.

Probabilistic Sharpe ratio

The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).

profit-loss ratio

The ratio of the average win rate to the average loss rate.

return

The rate of return across the entire trading period.

Sharpe ratio

A measure of the risk-adjusted return, developed by William Sharpe.

total fees

The total quantity of fees paid for all the transactions.

total net profit

The rate of return across the entire trading period.

total trades

The number of orders that were filled or partially filled.

tracking error

A measure of how closely a portfolio follows the index to which it is benchmarked. A tracking error of 0 is a perfect match.

Treynor ratio

A measurement of the returns earned in an algorithm in excess of the risk-free rate per unit of benchmark risk, developed by Jack Treynor.

unrealized

The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.

volume

The total value of assets traded for all of an algorithm's transactions.

win rate

The proportion of trades that were profitable after transaction fees.

Initialization

Introduction

The `Initialize` method is the entry point of your algorithm where you define a series of settings, including security subscriptions, starting cash balances, and warm-up periods. LEAN only calls the `Initialize` method one time, at the start of your algorithm.

Set Dates

To set the date range of backtests, call the `SetStartDate` and `SetEndDate` methods. The dates you provide are based in the algorithm time zone. By default, the end date is yesterday, one millisecond before midnight. In live trading, LEAN ignores the start and end dates.

```
PY
self.SetStartDate(2013, 1, 5)                  # Set start date to January 5, 2013
self.SetEndDate(2015, 1, 5)                    # Set end date to January 5, 2015
self.SetEndDate(datetime.now() - timedelta(7)) # Set end date to last week
```

Set Account Currency

The algorithm equity curve, benchmark, and performance statistics are denominated in the account currency. To set the account currency and your starting cash, call the `SetAccountCurrency` method. By default, the account currency is USD and your starting cash is $100,000. If you call the `SetAccountCurrency` method, you must call it before you call the `SetCash` method or add data . If you call the `SetAccountCurrency` method more than once, only the first call takes effect.

```
PY
self.SetAccountCurrency("BTC") # Set account currency to Bitcoin and its quantity to 100,000 BTC
self.SetAccountCurrency("INR") # Set account currency to Indian Rupees and its quantity to 100,000 INR
self.SetAccountCurrency("BTC", 10);  // Set account currency to Bitcoin and its quantity to 10 BTC
```

Set Cash

To set your starting cash in backtests, call the `SetCash` method. By default, your starting cash is $100,000 USD. In live trading, LEAN ignores the `SetCash` method and uses the cash balances in your brokerage account instead.

```
PY
self.SetCash(100000)       # Set the quantity of the account currency to 100,000
self.SetCash("BTC", 10)    # Set the Bitcoin quantity to 10
self.SetCash("EUR", 10000) # Set the EUR quantity to 10,000
```

Set Brokerage and Cash Model

We model your algorithm with margin modeling by default, but you can select a cash account type. Cash accounts don't allow leveraged trading, whereas Margin accounts can support leverage on your account value. To set your brokerage and account type, call the `SetBrokerageModel` method. For more information about each brokerage and the account

types they support, see the brokerage integration documentation. For more information about the reality models that the brokerage models set, see Supported Models .

```
self.SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Cash)
```

PY

The `AccountType` enumeration has the following members:

Set Universe Settings

The universe settings of your algorithm configure some properties of the universe constituents. The following table describes the properties of the `UniverseSettings` object:

Property: `ExtendedMarketHours` Should assets also feed extended market hours? You only receive extended market hours data if you create the subscription with an intraday resolution. If you create the subscription with daily resolution, the daily bars only reflect the regular trading hours. Data Type: `bool` | Default Value: `False`
Property: `FillForward` Should asset data fill forward? Data Type: `bool` | Default Value: `True`
Property: `MinimumTimeInUniverse` What's the minimum time assets should be in the universe? Data Type: `timedelta` | Default Value: `timedelta(1)`
Property: `Resolution` What resolution should assets use? Data Type: `Resolution` | Default Value: `Resolution.Minute`

Property: `ContractDepthOffset` What offset from the current front month should be used for continuous Future contracts ? 0 uses the front month and 1 uses the back month contract. This setting is only available for Future assets. Data Type: `int` | Default Value: `0`
Property: `DataMappingMode` How should continuous Future contracts be mapped? This setting is only available for Future assets. Data Type: `DataMappingMode` | Default Value: `DataMappingMode.OpenInterest`
Property: `DataNormalizationMode` How should historical prices be adjusted? This setting is only available for Equity and Futures assets. Data Type: `DataNormalizationMode` | Default Value: `DataNormalizationMode.Adjusted`
Property: `Leverage` What leverage should assets use in the universe? This setting is not available for derivative assets. Data Type: `float` | Default Value: `Security.NullLeverage`

To set the `UniverseSettings` , update the preceding properties in the `Initialize` method before you add the "universe" These settings are globals, so they apply to all universes you create.

```
# Request second resolution data. This will be slow!
self.UniverseSettings.Resolution = Resolution.Second
self.AddUniverse(self.MyCoarseFilterFunction)
```

Set Security Initializer

Instead of configuring global universe settings, you can individually configure the settings of each security in the universe with a security initializer. Security initializers let you apply any security-level reality model or special data requests on a per-security basis. To set the security initializer, in the `Initialize` method, call the `SetSecurityInitializer` method and then define the security initializer.

```py
#In Initialize
self.SetSecurityInitializer(self.CustomSecurityInitializer)

def CustomSecurityInitializer(self, security: Security) -> None:
    # Disable trading fees
    security.SetFeeModel(ConstantFeeModel(0, "USD"))
```

For simple requests, you can use the functional implementation of the security initializer. This style lets you configure the security object with one line of code.

```py
self.SetSecurityInitializer(lambda security: security.SetFeeModel(ConstantFeeModel(0, "USD")))
```

In some cases, you may want to trade a security in the same time loop that you create the security subscription. To avoid errors, use a security initializer to set the market price of each security to the last known price. The `GetLastKnownPrices` method seeds the security price by gathering the security data over the last 3 days. If there is no data during this period, the security price remains at 0.

```py
seeder = FuncSecuritySeeder(self.GetLastKnownPrices)
self.SetSecurityInitializer(lambda security: seeder.SeedSecurity(security))
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage
- Fee
- Buying Power
- Settlement
- Margin Interest Rate

The default security initializer also sets the leverage of each security and intializes each security with a seeder function. To extend upon the default security initializer instead of overwriting it, create a custom `BrokerageModelSecurityInitializer` .

```
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetFeeModel(ConstantFeeModel(0, "USD"))
```

To set a seeder function without overwriting the reality models of the brokerage, use the standard `BrokerageModelSecurityInitializer`.

```
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))
```

Add Data

You can subscribe to asset, fundamental, alternative, and custom data. The Dataset Market provides 400TB of data that you can easily import into your algorithms.

Asset Data

To subscribe to asset data, call one of the asset subscription methods like `AddEquity` or `AddForex`. Each asset class has its own method to create subscriptions. For more information about how to create subscriptions for each asset class, see Asset Classes.

```
self.AddEquity("SPY")  # Add Apple 1 minute bars (minute by default)
self.AddForex("EURUSD", Resolution.Second) # Add EURUSD 1 second bars
```

In live trading, you define the securities you want, but LEAN also gets the securities in your live portfolio and sets their resolution to the lowest resolution of the subscriptions you made. For example, if you create subscriptions in your algorithm for securities with Second, Minute, and Hour resolutions, the assets in your live portfolio are given a resolution of Second.

Alternative Data

To add alternative datasets to your algorithms, call the `AddData` method. For full examples, in the Datasets chapter, select a dataset and see the **Requesting Data** section.

Custom Data

To add custom data to your algorithms, call the `AddData` method. For more information about custom data, see Importing Data.

Limitations

There is no official limit to how much data you can add to your algorithms, but there are practical resource limitations. Each security subscription requires about 5MB of RAM, so larger machines let you run algorithms with bigger universes. For more information about our cloud nodes, see Resources .

Set Indicators and Consolidators

You can create and warm-up indicators in the `Initialize` method.

```
self.symbol = self.AddEquity("SPY").Symbol
self.sma = self.SMA(self.symbol, 20)
self.WarmUpIndicator(self.symbol, self.sma)
```

PY

Set Algorithm Settings

The following table describes the `AlgorithmSettings` properties:

Property: `DataSubscriptionLimit` The maximum number of concurrent market data subscriptions available. Data Type: `Int` | Default Value: `int.MaxValue`
Property: `FreePortfolioValue` The buying power buffer value. Data Type: `float` | Default Value: `250`
Property: `FreePortfolioValuePercentage` The buying power buffer percentage value. Data Type: `float` | Default Value: `0.0025`
Property: `LiquidateEnabled` A flag to enable and disable the `Liquidate` method. Data Type: `bool` | Default Value: `True`

Property: `MaxAbsolutePortfolioTargetPercentage`

The absolute maximum valid total portfolio value target percentage.

Data Type: `float` | Default Value: `1000000000`

Property: `MinAbsolutePortfolioTargetPercentage`

The absolute minimum valid total portfolio value target percentage.

Data Type: `float` | Default Value: `0.0000000001`

Property: `MinimumOrderMarginPortfolioPercentage`

The minimum order margin portfolio percentage to ignore bad orders and orders with small sizes.

Data Type: `float` | Default Value: `0.001`

Property: `RebalancePortfolioOnInsightChanges`

Rebalance the portfolio when you emit new insights or when insights expire.

Data Type: `bool/NoneType` | Default Value: `True`

Property: `RebalancePortfolioOnSecurityChanges`

Rebalance the portfolio when your universe changes.

Data Type: `bool/NoneType` | Default Value: `True`

Property: `StalePriceTimeSpan`

The minimum time span elapsed to consider a market fill price as stale

Data Type: `timedelta` | Default Value: `timedelta(hours=1)`

Property: `WarmUpResolution`

The resolution to use during the warm-up period

Data Type: `Resolution/NoneType` | Default Value: `None`

To change the `AlgorithmSettings` , update some of the preceding properties.

```
self.Settings.RebalancePortfolioOnSecurityChanges = False
```

To successfully update the `FreePortfolioValue` , you must update it after the `Initialize` method.

Set Benchmark

The benchmark performance is input to calculate several statistics on your algorithm, including alpha and beta . To set a benchmark for your algorithm, call the `SetBenchmark` method. You can set the benchmark to a security, a constant value, or a value from a custom data source . If you don't set a brokerage model , the default benchmark is SPY. If you set a brokerage model, the model defines the default benchmark.

```
# Set the benchmark to IBM
self.SetBenchmark("IBM")

# Set the benchmark to a constant value of 0
self.SetBenchmark(lambda x: 0)

# Set the benchmark to a value from a custom data source
self.symbol = self.AddData(CustomData, "CustomData", Resolution.Hour).Symbol
self.SetBenchmark(self.symbol)
```

If you pass a ticker to the `SetBenchmark` method, LEAN checks if you have a subscription for it. If you have a subscription for it, LEAN uses the security subscription. If you don't have a subscription for it, LEAN creates a US Equity subscription with the ticker. Since the ticker you pass may not reference a US Equity, we recommend you subscribe to the benchmark security before you call the `SetBenchmark` method.

Set Time Zone

LEAN supports international trading across multiple time zones and markets, so it needs a reference time zone for the algorithm to set the `Time` . The default time zone is Eastern Time (ET), which is UTC-4 in summer and UTC-5 in winter. To set a different time zone, call the `SetTimeZone` method. This method accepts either a string following the IANA Time Zone database convention or a NodaTime .DateTimeZone object. If you pass a string, the method converts it to a `NodaTime.DateTimeZone` object. The `TimeZones` class provides the following helper attributes to create `NodaTime.DateTimeZone` objects:

```
self.SetTimeZone("Europe/London")
self.SetTimeZone(TimeZones.Chicago)
```

The algorithm time zone may be different from the data time zone . If the time zones are different, it might appear like there is a lag between the algorithm time and the first bar of a history request, but this is just the difference in time zone. All the data is internally synchronized in Coordinated Universal Time (UTC) and arrives in the same Slice object. A slice is a sliver of time with all the data available for this moment.

To keep trades easy to compare between asset classes, we mark all orders in UTC time.

Set Warm Up Period

You may need some historical data at the start of your algorithm to prime technical indicators or populate historical data arrays. The warm-up period pumps data into your algorithm from before the start date. To set a warm-up period, call the `SetWarmUp` method. The warm-up feature uses the subscriptions you add in `Initialize` to gather the historical data that warms up the algorithm. If you don't create security subscriptions in the `Initialize` method, the warm-up won't occur.

```
PY
# Wind time back 7 days from the start date
self.SetWarmUp(timedelta(7))

# Feed in 100 trading days worth of data before the start date
self.SetWarmUp(100, Resolution.Daily)

# If you don't provide a resolution argument, it uses the lowest resolution in your subscriptions
self.SetWarmUp(100)
```

Post Initialization

After the `Initialize` method, the `PostInitialize` method performs post-initialization routines, so don't override it. To be notified when the algorithm is ready to begin trading, define an `OnWarmupFinished` method. This method executes even if you don't set a warm-up period.

```
PY
def OnWarmupFinished(self) -> None:
    self.Log("Algorithm Ready")
```

Examples

Demonstration Algorithms
BasicTemplateAlgorithm.py Python BasicTemplateCryptoAlgorithm.py Python BasicTemplateForexAlgorithm.py Python
BasicTemplateFuturesAlgorithm.py Python BasicTemplateOptionsAlgorithm.py Python

Securities

Securities

Key Concepts

Introduction

A security is an individual financial asset that you might trade on an exchange. LEAN models these unique assets with a `Security` object, which the `AddEquity` , `AddCrypto` , and similar methods return. The `Securities` property of the `QCAlgorithm` class is a dictionary where the keys are `Symbol` objects and the values are `Security` objects. The `Securities` dictionary contains all of the securities that have been in the algorithm during its lifetime.

Quote Currency

The quote currency is the currency you must give the seller to buy an asset. For currency trades, the quote currency is the currency ticker on the right side of the currency pair. For other types of assets, the quote currency is usually USD, but the quote currency for India Equities is INR. To get the quote currency of a `Security` , use the `QuoteCurrency` property.

```
aapl_quote_currency = self.Securities["AAPL"].QuoteCurrency # USD
btcusdt_quote_currency = self.Securities["BTCUSDT"].QuoteCurrency # USDT
```

The `QuoteCurrency` is a `Cash` object, which have the following properties:

You can use the `ConversionRate` property to calculate the value of the minimum price movement in the account currency

```
cfd = self.Securities["SG30SGD"]
quote_currency = cfd.QuoteCurrency # SGD
contract_mutiplier = cfd.SymbolProperties.ContractMultiplier
minimum_price_variation = cfd.SymbolProperties.MinimumPriceVariation

# Value of a pip in account currency
pip = minimum_price_variation * contract_mutiplier * quote_currency.ConversionRate
```

Security Listing

The `Exchange` property of a `Security` object contains information about the exchange that lists the security.

```
exchange = self.Securities["SPY"].Exchange
```

Security Market

The `Market` enumeration has the following members:

LEAN groups all of the US Equity exchanges under `Market.USA` . In live mode, the brokerage routes the orders to the exchange that provides the best price.

Active Securities

The `ActiveSecurities` property of the algorithm class contains all of the assets currently in your universe. It is a dictionary where the key is a `Symbol` and the value is a `Security` . When you remove an asset from a universe, LEAN usually removes the security from the `ActiveSecurities` collection and removes the security subscription. However, it won't remove the security in any of the following situations:

- You own the security.
- You have an open order for the security.
- The security wasn't in the universe long enough to meet the `MinimumTimeInUniverse` setting.

When LEAN removes the security, the `Security` object remains in the `Securities` collection for record-keeping purposes, like tracking fees and trading volume.

Tradable Status

The `IsTradable` property shows whether you can trade a security. The property value is true when the security is in the universe, even if the data starts at a later day. Indices, canonical Option securities, and continuous Futures contracts are not tradable. In live mode, custom data objects are also not tradable, even if the custom data represents a tradable asset.

```
tradable = self.Securities["SPY"].IsTradable
```
PY

Linked Custom Data

Linked custom data is custom data that's linked to individual assets. In contrast, unlinked data is not linked to specific assets. An example of an unlinked dataset is the US Regulatory Alerts dataset. LEAN passes custom data objects to the `OnData` method. For more information about custom data, see Custom Securities .

Security Types

LEAN uses the `SecurityType` enumeration as a synonym for asset classes. The `SecurityType` enumeration has the following members:

The `Base` type is for non-financial assets like custom and alternative data objects.

Securities

Properties

Introduction

`Security` objects contain the models and properties of an asset.

Security Properties

`Security` objects have the following properties:

Reality Models

`Security` objects contain references to the security level reality models . To customize the behavior of each security, configure its reality models.

Symbol Properties

The `SymbolProperties` property of a `Security` contains properties for a specific security. `SymbolProperties` objects have the following properties:

To create a `SymbolProperties` object, call the constructor.

```
symbol_properties = SymbolProperties(description, quoteCurrency, contractMultiplier,
minimumPriceVariation, lotSize, marketTicker)
```

PY

The following table describes the arguments of the `SymbolProperties` constructor:

Argument	Data Type	Description	Default Value
`description`	`string str`	The description of the security.	
`quoteCurrency`	`string str`	The quote currency of the security.	
`contractMultiplier`	`decimal float`	The contract multiplier for the security.	
`minimumPriceVariation`	`decimal float`	The minimum price variation (tick size) for the security.	
`lotSize`	`decimal float`	The lot size (lot size of the order) for the security.	
`marketTicker`	`string str`	The market ticker.	
`minimumOrderSize`	`decimal? float/NoneType`	The minimum order size allowed.	`None`
`priceMagnifier`	`decimal float`	This property allows normalizing live asset prices to US Dollars for Lean consumption. In some exchanges, for some securities, data is expressed in cents like corn Futures ('ZC').	1

Securities

Exchange

Introduction

The `Exchange` property of a `Security` object contains information about the exchange that lists the security.

Properties

The `Exchange` property of a `Security` object returns an `SecurityExchange` object, which has the following attributes:

Hours

To get the exchange hours, use the `Exchange.Hours` property on a `Security` object.

```
security_exchange_hours = self.Securities["SPY"].Exchange.Hours
```

The preceding code snippet returns a `SecurityExchangeHours` object, which has the following attributes:

To check if the exchange is open at a specific time, call the `IsOpen` method.

```
# Check if the exchange is open at a specific time
is_open_now = security_exchange_hours.IsOpen(self.Time, extendedMarketHours=False)

# Check if the exchange is open at any point in time over a specific interval
is_open = security_exchange_hours.IsOpen(self.Time, self.Time + timedelta(days=1),
extendedMarketHours=False)
```

To check if the exchange is open on a specific date, call the `IsDateOpen` method.

```
is_open = security_exchange_hours.IsDateOpen(self.Time)
```

To get the next market open time, call the `GetNextMarketOpen` method.

```
market_open_time = security_exchange_hours.GetNextMarketOpen(self.Time, extendedMarketHours=False)
```

To get the next market close time, call the `GetNextMarketClose` method.

```
market_close_time = security_exchange_hours.GetNextMarketClose(self.Time, extendedMarketHours=False)
```

To get the previous trading day, call the `GetPreviousTradingDay` method.

```
day = security_exchange_hours.GetPreviousTradingDay(self.Time)
```

To get the next trading day, call the `GetNextTradingDay` method.

```
day = security_exchange_hours.GetNextTradingDay(self.Time)
```

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.AddEquity("SPY", extendedMarketHours=True)
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

24 Hour Markets

The Crypto market trades 24/7, but other asset classes only trade during part of the week. Each market has an official trading schedule, but it can be impacted by daylight savings, holidays, and trading halts.

Securities

Requesting Data

Introduction

If you create a data subscription, your algorithm receives data updates for that security or custom data source.

Characteristics

We thoroughly vet datasets before we add them to the Data Market. The datasets in the Data Market are well-defined with robust ticker and `Symbol` links to ensure tickers are properly tracked through time. The datasets are consistently delivered on time and most of them are ready for live trading. All the datasets have at least one year of historical data and are free of survivorship bias.

Resolutions

Resolution is the duration of time that's used to sample a data source. The `Resolution` enumeration has the following members:

The default resolution for market data is `Minute` . To set the resolution for a security, pass the `resolution` argument when you create the security subscription.

```
self.AddEquity("SPY", Resolution.Daily)
```

PY

To set the resolution for all securities, set the `Resolution` universe setting before you create security subscriptions.

```
self.UniverseSettings.Resolution = Resolution.Daily
```

PY

To see which resolutions of data are available for a dataset, see the dataset listing in the Data Market . To create custom resolution periods, see Consolidating Data .

Data density describes the frequency of entries in a dataset. Datasets at the tick resolution have dense data density. All other resolutions usually have regular data density. If a non-tick resolution dataset doesn't have an entry at each sampling, it has sparse density.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.AddEquity("SPY", fillForward=False)
```

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.AddEquity("SPY", extendedMarketHours=True)
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

Remove Subscriptions

To remove a security subscription, call the `RemoveSecurity` method.

```
self.RemoveSecurity(self.symbol)
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings.

Markets

The datasets integrated into the Dataset Market cover many markets. The `Market` enumeration has the following members:

LEAN can usually determine the correct market based on the ticker you provide when you create the security subscription. To manually set the market for a security, pass a `market` argument when you create the security subscription.

```
self.AddEquity("SPY", market=Market.USA)
```

Quotas

There is no limit to the number of security subscriptions you can create, but some live data providers impose data subscription limits. To set a limit, in the `Initialize` method, set the `DataSubscriptionLimit` algorithm setting .

```
self.Settings.DataSubscriptionLimit = 500
```

All securities that you add to the algorithm count as one subscription, except Options and Futures. Every Option and Future contract counts as one subscription.

Securities

Handling Data

Introduction

LEAN packages the data for your subscriptions in a `Slice` object and passes it to the `OnData` method of your algorithm. To avoid look-ahead bias , LEAN only provides the data that's available at the current time in your algorithm. This processing style ensures that your algorithm backtests in the same manner that it trades live. Your algorithm uses the data to make trading decisions and LEAN uses the data to update your positions, track your portfolio value, and simulate orders.

Security Cache

To get the most recent data for a security, call the `GetLastData` method on the `Security` object.

```
PY
data = self.Securities["SPY"].GetLastData()
```

Timeslice

The `Slice` that LEAN passes to the `OnData` method represents all of the data for your subscriptions at a single point in time. The `Slice` object contains data like `Tick` objects, `TradeBar` objects, `QuoteBar` objects, corporate actions , and chains for Option and Future contracts. You can use the data in the `Slice` to make trading decisions.

To access data in the `Slice` , index it with the security `Symbol` . If you use the security ticker instead of the `Symbol` , LEAN automatically finds the `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.symbol):
        my_data = slice[self.symbol]
```

The following table shows the `Slice` property to index based on the data format you want:

Data Format	Slice Property
`TradeBar`	`Bars`
`QuoteBar`	`QuoteBars`
`Tick`	`Ticks`

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.symbol):
        bar = slice.Bars[self.symbol]
```

If you just index the `Slice` instead of the preceding properties, it returns the correct object. If your data subscription provides `QuoteBar` objects and you index the `Slice` with the security `Symbol` , it returns the `QuoteBar` .

`Slice` objects have the following properties:

For more information about the `Slice` class, see Timeslices .

Securities

Filtering Data

Introduction

Unfiltered raw data can be faulty for a number of reasons, including invalid data entry. Moreover, high-frequency traders can deploy bait-and-switch strategies by submitting bait orders to deceive other market participants, making raw data noisy and untradeable. To avoid messing up with our trading logic and model training, you can filter out suspicious raw data with a data filter.

Set Models

To set a data filter for a security, call the `SetDataFilter` property on the `Security` object.

```py
# In Initialize
spy = AddEquity("SPY")
spy.SetDataFilter(SecurityDataFilter())
```

You can also set the data filter model in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

```py
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetDataFilter(SecurityDataFilter())
```

Default Behavior

The following table shows the default data filter for each security type:

Security Type	Default Filter
`Equity`	EquityDataFilter
`Option`	OptionDataFilter
`Forex`	ForexDataFilter
`Index`	IndexDataFilter
`Cfd`	CfdDataFilter
Others	SecurityDataFilter

None of the preceding filters filter out any data.

Model Structure

Data filtering models must implement a `Filter` method, which receives `Security` and `BaseData` objects and then returns a `boolean` object that represents if the data point should be filtered out.

```
class MyDataFilter:
    def Filter(self, vehicle: Security, data: BaseData) -> bool:
        return True
```

Examples

Demonstration Algorithms
CustomSecurityDataFilterRegressionAlgorithm.py Python TickDataFilteringAlgorithm.py Python

Securities

Asset Classes

You can trade any of the following asset classes. Click one to learn more.

US Equity

India Equity

Equity Options

Crypto

Crypto Futures

Forex

Futures

Future Options

Index

Index Options

CFD

See Also

Universes
Importing Data

Asset Classes

US Equity

US Equities represent partial ownership in a US corporation.

Requesting Data

Handling Data

Corporate Actions

Corporate Fundamentals

Shorting

Data Preparation

Market Hours

See Also

BasicTemplateAlgorithm.py
BasicTemplateAlgorithm.cs

US Equity

Requesting Data

Introduction

Request US Equity data in your algorithm to receive a feed of asset prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Equities dataset listing . To trade US Equities live, you can use our US Equities data feed or one of the brokerage data feeds .

Create Subscriptions

To create an Equity subscription, in the `Initialize` method, call the `AddEquity` method. The `AddEquity` method returns an `Equity` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
self.symbol = self.AddEquity("SPY").Symbol
```

The `AddEquity` method creates a subscription for a single Equity asset and adds it to your **user-defined** universe. To create a dynamic universe of Equities, add an Equity universe or an Equity Universe Selection model .

To view the supported assets in the US Equities dataset, see the Data Explorer .

Resolutions

The following table shows the available resolutions and data formats for Equity subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓			
`Daily`	✓			

The default resolution for Equity subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddEquity` method.

```
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

To create custom resolution periods, see Consolidating Data .

Asset Primary Exchange

When a stock like Apple is listed, it's listed on Nasdaq. The open auction tick on Nasdaq is the price that's used as the official open of the day. NYSE, BATS, and other exchanges also have opening auctions, but the only official opening price for Apple is the opening auction on the exchange where it was listed.

Supported Markets

LEAN groups all of the US Equity exchanges under `Market.USA` . In live mode, the brokerage routes the orders to the exchange that provides the best price.

To set the market for a security, pass a `market` argument to the `AddEquity` method.

```
self.symbol = self.AddEquity("SPY", market=Market.USA).Symbol
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.AddEquity("SPY", fillForward=False)
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. In backtests, the default leverage for margin accounts is 2x leverage and leverage is not available for cash accounts. To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddEquity` method.

```
self.symbol = self.AddEquity("SPY", leverage=3).Symbol
```

In live trading, the brokerage determines how much leverage you may use. For more information about the leverage they provide, see Brokerages .

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.AddEquity("SPY", extendedMarketHours=True)
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If

you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Data Normalization

The data normalization mode defines how historical data is adjusted for corporate actions . The data normalization mode affects the data that LEAN passes to `OnData` and the data from history requests . By default, LEAN adjusts US Equity data for splits and dividends to produce a smooth price curve, but the following data normalization modes are available:

We use the entire split and dividend history to adjust historical prices. This process ensures you get the same adjusted prices, regardless of the backtest end date.

To set the data normalization mode for a security, pass a `dataNormalizationMode` argument to the `AddEquity` method.

```
self.symbol = self.AddEquity("SPY", dataNormalizationMode=DataNormalizationMode.Raw).Symbol
```

PY

Properties

The `AddEquity` method returns an `Equity` object, which have the following properties:

US Equity

Handling Data

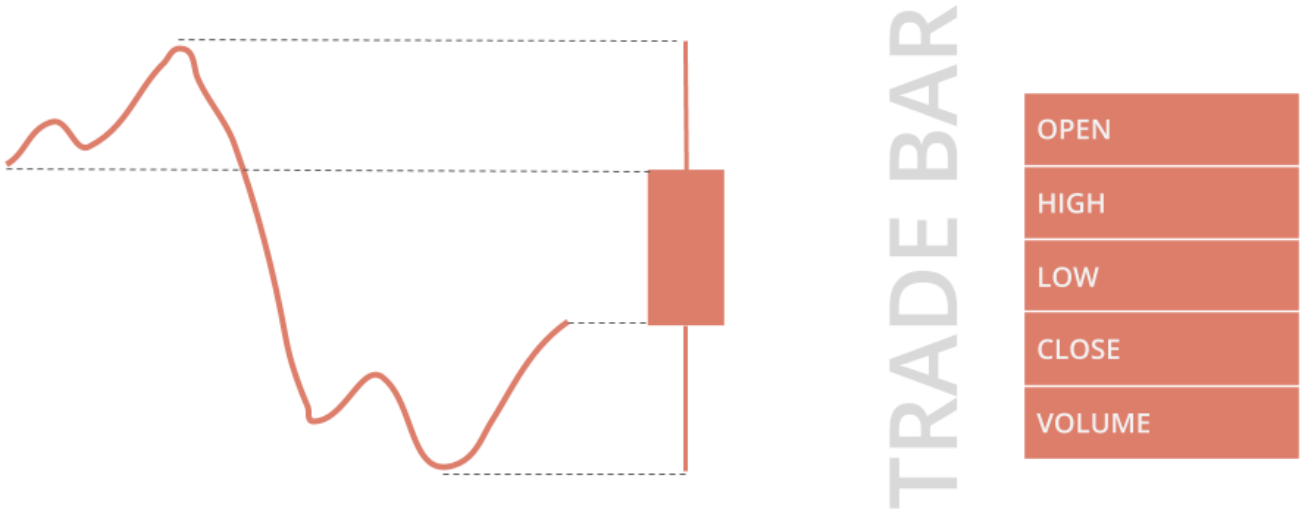
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
```

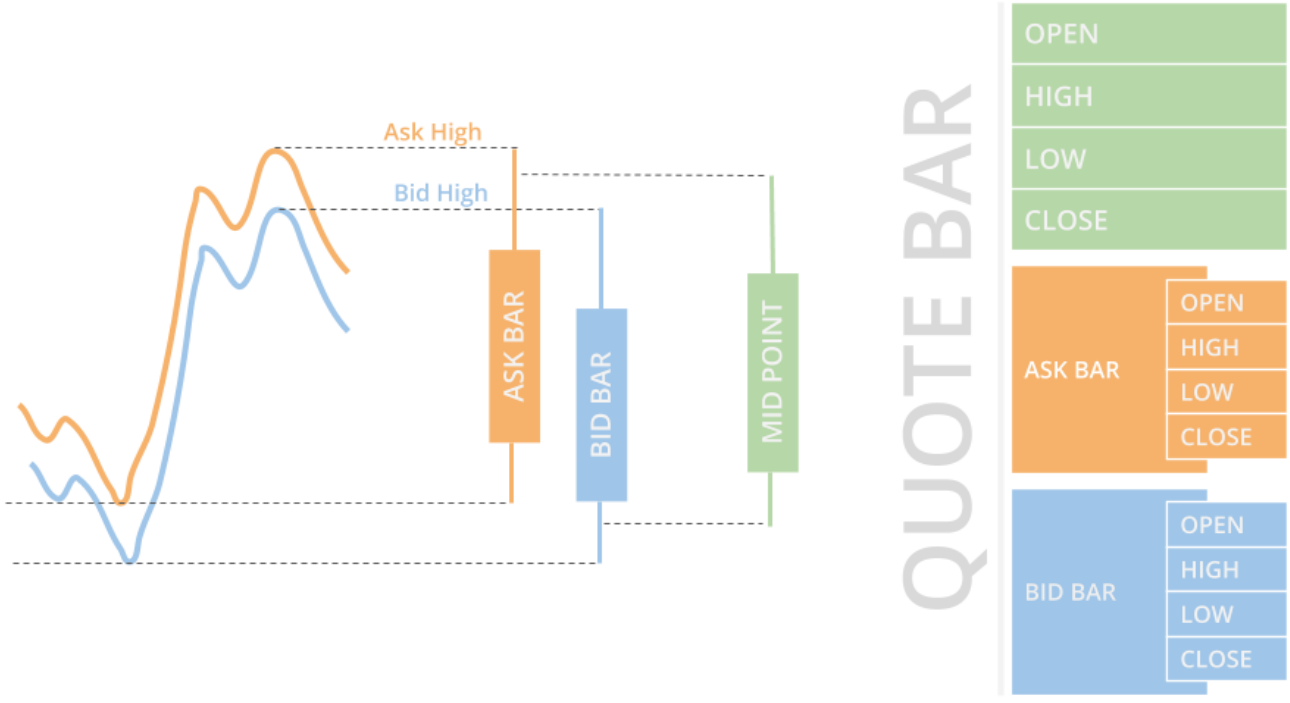
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

We adjust the daily open and close price of bars to reflect the official auction prices .

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the security `Symbol` . If the security doesn't actively get quotes or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the security. A quote tick is an offer to buy or sell the security at a specific price. `Tick` objects have the following properties:

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            price = tick.Price
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `list[Tick]` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            price = tick.Price
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able

to perform your own online tick filtering.

If we detect a tick that may be suspicious , we set its `Suspicious` flag to true.

Other Data Formats

For more information about data formats available for US Equities, see Corporate Actions .

US Equity

Corporate Actions

Introduction

US Equity subscriptions provide notifications for splits, dividends, symbol changes, and delistings.

Splits

When a company does a stock split, the number of shares each shareholder owns increases and the price of each share decreases. When a company does a reverse stock split, the number of shares each shareholder owns decreases and the price of each share increases. A company may perform a stock split or a reverse stock split to adjust the price of their stock so that more investors trade it and the liquidity increases.

When a stock split or reverse stock split occurs for an Equity in your algorithm, LEAN sends a `Split` object to the `OnData` method. Split objects have the following properties:

You receive `Split` objects when a split is in the near future and when it occurs. To know if the split occurs in the near future or now, check the `Type` property.

If you backtest without the `Raw` data normalization mode , the splits are factored into the price and volume. If you backtest with the `Raw` data normalization mode or trade live, when a split occurs, LEAN automatically adjusts your positions based on the `SplitFactor` . If the post-split quantity isn't a valid lot size , LEAN credits the remaining value to your cashbook in your account currency. If you have indicators in your algorithm, reset your indicators when splits occur so that the data in your indicators account for the price adjustments that the splits cause.

To get the `Split` objects in the `Slice` , index the `Splits` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Splits` property contains data for your security before you index it with the security `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    split = slice.Splits.get(self.symbol)
    if split:
        pass
```

You can also iterate through the `Splits` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Split` objects.

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, split in slice.Splits.items():
        pass
```

LEAN stores the data for stock splits in factor files. To view some example factor files, see the LEAN GitHub repository . In backtests, your algorithm receives `Split` objects at midnight. In live trading, your algorithm receives `Split` objects

when the factor files are ready.

If you hold an Option contract for an underlying Equity when a split occurs, LEAN closes your Option contract position.

If a split event occurs before your order is filled, the unfilled portion of the order is adjusted automatically, where its quantity is multiplied by the split factor and the limit/stop/trigger price (if any) is divided by the split factor.

Dividends

A dividend is a payment that a company gives to shareholders to distribute profits. When a dividend payment occurs for an Equity in your algorithm, LEAN sends a `Dividend` object to the `OnData` method. `Dividend` objects have the following properties:

If you backtest with the `Adjusted` or `TotalReturn` data normalization mode, the dividends are factored into the price. If you backtest with the other data normalization modes or trade live, when a dividend payment occurs, LEAN automatically adds the payment amount to your cashbook. If you have indicators in your algorithm, reset your indicators when dividend payments occur so that the data in your indicators account for the price adjustments that the dividend causes.

To get the `Dividend` objects in the `Slice` , index the `Dividends` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Dividends` property contains data for your security before you index it with the security `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    dividend = slice.Dividends.get(self.symbol)
    if dividend:
        pass
```

You can also iterate through the `Dividends` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Dividend` objects.

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, dividend in slice.Dividends.items():
        pass
```

For a full example, see the DividendAlgorithm in the LEAN GitHub repository.

Symbol Changes

The benefit of the `Symbol` class is that it always maps to the same security, regardless of their trading ticker. When a company changes its trading ticker, LEAN sends a `SymbolChangedEvent` to the `OnData` method. `SymbolChangedEvent` objects have the following properties:

To get the `SymbolChangedEvent` objects in the `Slice` , index the `SymbolChangedEvents` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `SymbolChangedEvents` property contains data for your security before you index it with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    symbol_changed_event = slice.SymbolChangedEvents.get(self.symbol)
    if symbol_changed_event:
        pass
```

You can also iterate through the `SymbolChangedEvents` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `SymbolChangedEvent` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, symbol_changed_event in slice.SymbolChangedEvents.items():
        pass
```

If you have an open order for a security when they change their ticker, LEAN cancels your order. To keep your order, in the `OnOrderEvent` method, get the quantity and `Symbol` of the cancelled order and submit a new order.

```
def OnOrderEvent(self, order_event: OrderEvent) -> None:
    if order_event.Status == OrderStatus.Canceled:
        ticket = self.Transactions.GetOrderTicket(order_event.OrderId)
        if "symbol changed event" in ticket.Tag:
            self.Transactions.AddOrder(ticket.SubmitRequest)
```

LEAN stores the data for ticker changes in map files. To view some example map files, see the LEAN GitHub repository .

Delistings

When a company is delisting from an exchange, LEAN sends a `Delisting` object to the `OnData` method. `Delisting` objects have the following properties:

You receive `Delisting` objects when a delisting is in the near future and when it occurs. To know if the delisting occurs in the near future or now, check the `Type` property.

To get the `Delisting` objects in the `Slice` , index the `Delistings` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Delistings` property contains data for your security before you index it with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    delisting = slice.Delistings.get(self.symbol)
    if delisting:
        pass
```

You can also iterate through the `Delistings` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Delisting` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, delisting in slice.Delistings.items():
        pass
```

The delist warning occurs on the final trading day of the stock to give you time to gracefully exit out of positions before LEAN automatically liquidates them.

```py
if delisting.Type == DelistingType.Warning:
    # Liquidate with MarketOnOpenOrder on delisting warning
    quantity = self.Portfolio[symbol].Quantity
    if quantity != 0:
        self.MarketOnOpenOrder(symbol, -quantity)
```

For a full example, see the DelistingEventsAlgorithm in the LEAN GitHub repository.

US Equity

Corporate Fundamentals

Introduction

Corporate fundamental data contains all the information on the underlying company of an Equity asset and the information in their financial statements. Since corporate data contains information not found in price and alternative data, adding corporate data to your trading strategies provides you with more information so you can make more informed trading decisions. Corporate fundamental data is available through the US Fundamental Data from Morningstar . To get fundamental data into your algorithm, add a fundamental universe .

Properties

To get fundamental data, access the `FineFundamental` properties in your fine universe selection function or access the `Fundamentals` property of the `Equity` objects in your fundamental universe.

```
PY
fundamentals = self.Securities[self.symbol].Fundamentals
```

The US Fundamentals dataset provides FineFundamental objects. To filter FineFundamental objects, you can use the MorningstarSectorCode, MorningstarIndustryGroupCode, and MorningstarIndustryCode enumeration values.

FineFundamental Attributes

FineFundamental objects have the following attributes:

MorningstarSectorCode Enumeration

Sectors are large super categories of data. To access the sector of an Equity, use the **MorningstarSectorCode** property.

```
PY
filtered_fine = [x for x in fine if x.AssetClassification.MorningstarSectorCode ==
MorningstarSectorCode.Technology]
```

The **MorningstarSectorCode** enumeration has the following members:

MorningstarIndustryGroupCode Enumeration

Industry groups are clusters of related industries which tie together. To access the industry group of an Equity, use the **MorningstarIndustryGroupCode** property.

```
PY
filtered_fine = [x for x in fine if x.AssetClassification.MorningstarIndustryGroupCode ==
MorningstarIndustryGroupCode.ApplicationSoftware]
```

The **MorningstarIndustryGroupCode** enumeration has the following members:

MorningstarIndustryCode Enumeration

Industries are the finest level of classification available and are the individual industries according to the Morningstar classification system. To access the industry group of an Equity, use the **MorningstarIndustryCode** property:

```
PY
filtered_fine = [x for x in fine if x.AssetClassification.MorningstarIndustryCode ==
MorningstarIndustryCode.SoftwareApplication]
```

The **MorningstarIndustryCode** enumeration has the following members:

Exchange Id Values

Exchange Id is mapped to represent the exchange that lists the Equity. To access the exchange Id of an Equity, use the **PrimaryExchangeID** property.

```
PY
filtered_fine = [x for x in fine if x.CompanyReference.PrimaryExchangeID == "NAS"]
```

The exchanges are represented by the following string values:

String Representation	Exchange
NYS	New York Stock Exchange (NYSE)
NAS	NASDAQ
ASE	American Stock Exchange (AMEX)
TSE	Tokyo Stock Exchange
AMS	Amsterdam Internet Exchange
SGO	Santiago Stock Exchange
XMAD	Madrid Stock Exchange
ASX	Australian Securities Exchange
BVMF	B3 (stock exchange)
LON	London Stock Exchange
TKS	Istanbul Stock Exchange Settlement and Custody Bank
SHG	Shanghai Exchange
LIM	Lima Stock Exchange

String Representation	Exchange
FRA	Frankfurt Stock Exchange
JSE	Johannesburg Stock Exchange
MIL	Milan Stock Exchange
TAE	Tel Aviv Stock Exchange
STO	Stockholm Stock Exchange
ETR	Deutsche Boerse Xetra Core
PAR	Paris Stock Exchange
BUE	Buenos Aires Stock Exchange
KRX	Korea Exchange
SWX	SIX Swiss Exchange
PINX	Pink Sheets (OTC)
CSE	Canadian Securities Exchange
PHS	Philippine Stock Exchange
MEX	Mexican Stock Exchange
TAI	Taiwan Stock Exchange
IDX	Indonesia Stock Exchange
OSL	Oslo Stock Exchange
BOG	Colombia Stock Exchange
NSE	National Stock Exchange of India
HEL	Nasdaq Helsinki
MISX	Moscow Exchange
HKG	Hong Kong Stock Exchange
IST	Istanbul Stock Exchange
BOM	Bombay Stock Exchange
TSX	Toronto Stock Exchange
BRU	Brussels Stock Exchange

String Representation	Exchange
BATS	BATS Global Markets
ARCX	NYSE Arca
GREY	Grey Market (OTC)
DUS	Dusseldorf Stock Exchange
BER	Berlin Stock Exchange
ROCO	Taipei Exchange
CNQ	Canadian Trading and Quotation System Inc.
BSP	Bangko Sentral ng Pilipinas
NEOE	NEO Exchange

Historical Data

To get historical fundamental data, set a warm-up period and save the fundamental data during the warm-up period. It's not currently possible to make a history request for fundamental data in an algorithm. Subscribe to GitHub Issue #4890 to track the feature progress.

Data Updates

The US Fundmamental dataset only provides the originally-reported figures. If there was a mistake in reporting a figure, the data value isn't fixed later. In live trading, new fundamental data is available to your algorithms at approximately 6 AM Eastern Time (ET) each day. The majority of the corporate data update occurs once per month, but the financial ratios update daily. If there is no new data for a period of time, the previous data is filled forward.

US Equity

Shorting

Introduction

In a regular long position, you buy shares and then sell them later to close the trade. In a short position, you borrow shares, sell them, and then buy the shares back later to close the trade. Short positions let you profit when a security's price decreases while you have the position open, but there are risks involved. For example, you can get a margin call and you may incur borrowing costs.

Holdings Accounting

If you have an open long position, your portfolio has a positive quantity of shares for that security. In contrast, if you have an open short position, your portfolio has a negative quantity of shares for that security.

Borrowing Costs

In live trading, you usually pay a fee to borrow shares that you use to open a short position. Part of the fee goes to the investor that provided you with the opportunity to short with their shares. The borrowing rate is set by your live brokerage. In backtests, LEAN doesn't currently model borrowing costs, but we have an open GitHub Issue to add the functionality. Subscribe to GitHub Issue #4563 to track the feature progress.

US Equity

Data Preparation

Introduction

The US Equities dataset provides price data for backtests and live trading.

Sourcing

The US Equities data feed consolidates market data across all of the exchanges. Over-the-Counter (OTC) trades are excluded. The data feed is powered by the Securities Information Processor (SIP), so it has 100% market coverage. In contrast, free platforms that display data feeds like the Better Alternative Trading System (BATS) only have about 6-7% market coverage .

We provide live splits, dividends, and corporate actions for US companies. We deliver them to your algorithm before the trading day starts.

Bar Building

We aggregate ticks to build bars.

The bar-building process can exclude ticks. If a tick is excluded, its volume is aggregated in the bar but its price is not aggregated in the bar. Ticks are excluded if any of the following statements are true:

- The tick is suspicious.
- The tick is from the FINRA exchange and meets our price and volume thresholds.
- The trade has none of the following included `TradeConditionFlags` and at least one of the following excluded `TradeConditionFlags` :

`TradeConditionFlags`	**Status**	**Description**
`Regular`	Included	A trade made without stated conditions is deemed the regular way for settlement on the third business day following the transaction date.
`FormT`	Included	Trading in extended hours enables investors to react quickly to events that typically occur outside regular market hours, such as earnings reports. However, liquidity may be constrained during such Form T trading, resulting in wide bid-ask spreads.
`Cash`	Included	A transaction that requires delivery of securities and payment on the same day the trade takes place.

TradeConditionFlags	Status	Description
ExtendedHours	Included	Identifies a trade that was executed outside of regular primary market hours and is reported as an extended hours trade.
NextDay	Included	A transaction that requires the delivery of securities on the first business day following the trade date.
OfficialClose	Included	Indicates the "official" closing value determined by a Market Center. This transaction report will contain the market center generated closing price.
OfficialOpen	Included	Indicates the 'Official' open value as determined by a Market Center. This transaction report will contain the market center generated opening price.
ClosingPrints	Included	The transaction that constituted the trade-through was a single priced closing transaction by the Market Center.
OpeningPrints	Included	The trade that constituted the trade-through was a single priced opening transaction by the Market Center.
IntermarketSweep	Excluded	The transaction that constituted the trade-through was the execution of an order identified as an Intermarket Sweep Order.
TradeThroughExempt	Excluded	Denotes whether or not a trade is exempt (Rule 611).
OddLot	Excluded	Denotes the trade is an odd lot less than a 100 shares.

- The quote has a size of less than 100 shares.
- The quote has one of the following `QuoteConditionFlags` :

QuoteConditionFlags	Description
Closing	Indicates that this quote was the last quote for a security for that Participant.
NewsDissemination	Denotes a regulatory trading halt when relevant news influencing the security is being disseminated. Trading is suspended until the primary market determines that an adequate publication or disclosure of information has occurred.
NewsPending	Denotes a regulatory Trading Halt due to an expected news announcement, which may influence the security. An Opening Delay or Trading Halt may be continued once the news has been disseminated.
TradingRangeIndication	Denotes the probable trading range (Bid and Offer prices, no sizes) of a security that is not Opening Delayed or Trading Halted. The Trading Range Indication is used prior to or after the opening of a security.
OrderImbalance	Denotes a non-regulatory halt condition where there is a significant imbalance of buy or sell orders.
Resume	Indicates that trading for a Participant is no longer suspended in a security that had been Opening Delayed or Trading Halted.

- The quote has none of the following `QuoteConditionFlags` :

QuoteConditionFlags	Description
Regular	This condition is used for the majority of quotes to indicate a normal trading environment.
Slow	This condition is used to indicate that the quote is a Slow Quote on both the bid and offer sides due to a Set Slow List that includes high price securities.
Gap	While in this mode, auto-execution is not eligible, the quote is then considered manual and non-firm in the bid and offer, and either or both sides can be traded through as per Regulation NMS.
OpeningQuote	This condition can be disseminated to indicate that this quote was the opening quote for a security for that Participant.
FastTrading	For extremely active periods of short duration. While in this mode, the UTP Participant will enter quotations on a best efforts basis.
Resume	Indicate that trading for a Participant is no longer suspended in a security which had been Opening Delayed or Trading Halted.

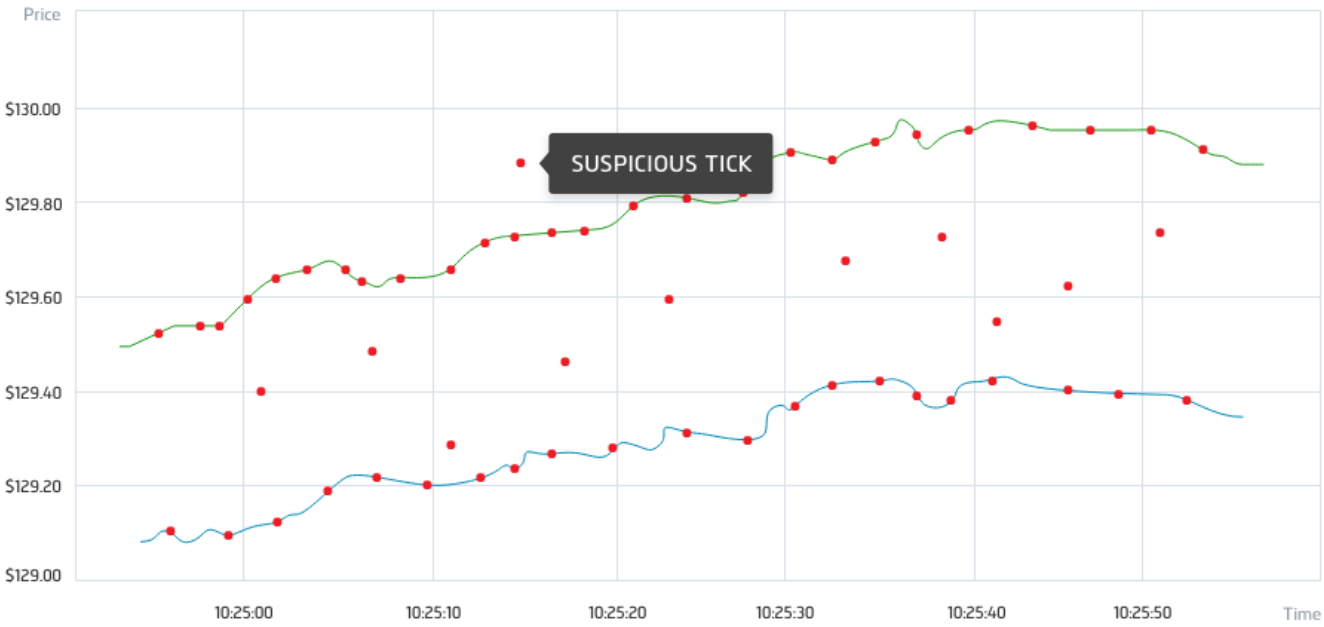
In the preceding tables, **Participant** refers to the entities on page 19 of the Consolidated Tape System Multicast Output Binary Specification .

Suspicious Ticks

Tick price data is raw and unfiltered, so it can contain a lot of noise. If a tick is not tradable, we flag it as suspicious. This process makes the bars a more realistic representation of what you could execute in live trading. If you use tick data, avoid using suspicious ticks in your algorithms as informative data points. We recommend only using tick data if you understand the risks and are able to perform your own tick filtering. Ticks are flagged as suspicious in the following

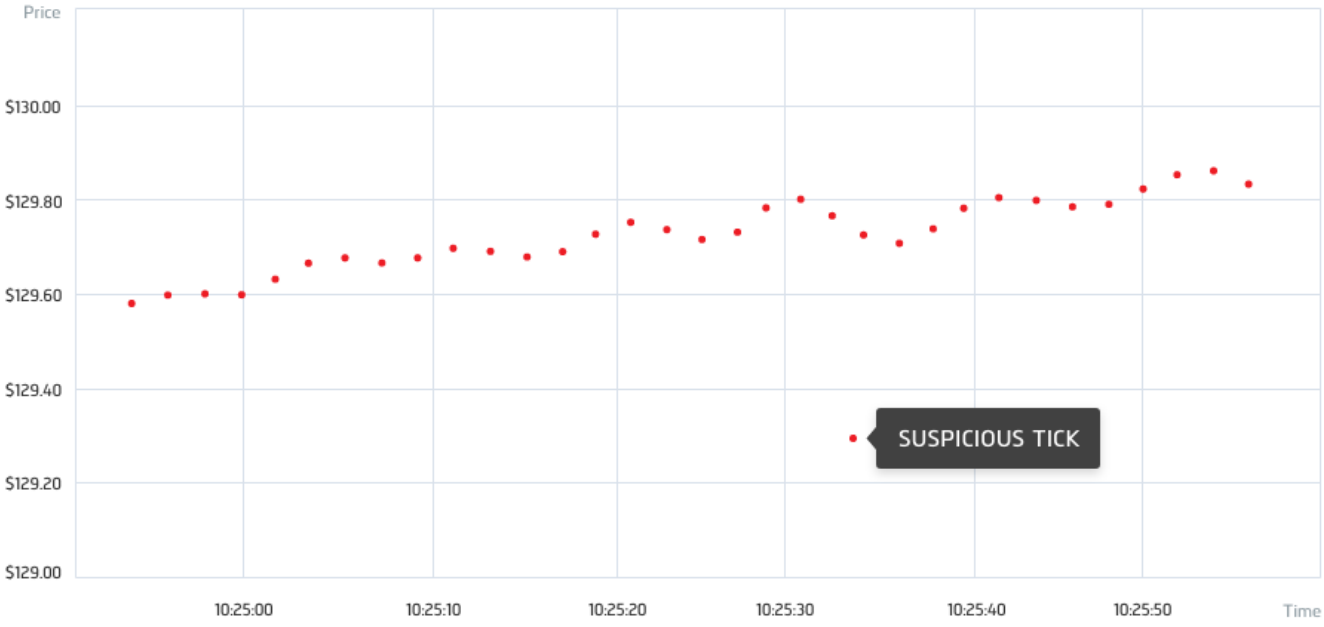
situations:

- The tick occurs below the best bid or above the best ask



This image shows a tick that occurred above the best ask price of a security. The green line represents the best ask of the security, the blue line represents the best bid of the security, and the red dots represent trade ticks. The ticks between the best bid and ask occur from filling hidden orders. The tick that occurred above the best ask price is flagged as suspicious.

- The tick occurs far from the current market price



This image shows a tick that occurred far from the price of the security. The red dots represent trade ticks. The tick that occurred far from the market price is flagged as suspicious.

- The tick occurs on a dark pool
- The tick is rolled back

- The tick is reported late

Market Auction Prices

The opening and closing price of the day is set by very specific opening and closing auction ticks. When a stock like Apple is listed, it's listed on Nasdaq. The open auction tick on Nasdaq is the price that's used as the official open of the day. NYSE, BATS, and other exchanges also have opening auctions, but the only official opening price for Apple is the opening auction on the exchange where it was listed.

We set the opening and closing prices of the first and last bars of the day to the official auction prices. This process is used for second, minute, hour, and daily bars for the 9:30 AM and 4:30 PM Eastern Time (ET) prices. In contrast, other platforms might not be using the correct opening and closing prices.

The official auction prices are usually emitted 2-30 seconds after the market open and close. We do our best to use the official opening and closing prices in the bars we build, but the delay can be so large that there isn't enough time to update the opening and closing price of the bar before it's injected into your algorithms. For example, if you subscribe to second resolution data, we wait until the end of the second for the opening price but most second resolution data won't get the official opening price. If you subscribe to minute resolution data, we wait until the end of the minute for the opening auction price. Most of the time, you'll get the actual opening auction price with minute resolution data, but there are always exceptions. Nasdaq and NYSE can have delays in publishing the opening auction price, but we don't have control over those issues and we have to emit the data on time so that you get the bar you are expecting.

Live and Backtesting Differences

In live trading, bars are built using the exchange timestamps with microsecond accuracy. This microsecond-by-microsecond processing of the ticks can mean that the individual bars between live trading and backtesting can have slightly different ticks. As a result, it's possible for a tick to be counted in different bars between backtesting and live trading, which can lead to bars having slightly different open, high, low, close, and volume values.

Data Availability

In live trading, live data is available in real-time. In backtests, live data is available at the following pre-market trading session.

US Equity

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the US Equity market.

Pre-market Hours

The following table shows the pre-market hours for the US Equity market:

Weekday	Time (America/New York)
Monday	04:00:00 to 09:30:00
Tuesday	04:00:00 to 09:30:00
Wednesday	04:00:00 to 09:30:00
Thursday	04:00:00 to 09:30:00
Friday	04:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the US Equity market:

Weekday	Time (America/New York)
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the US Equity market:

Weekday	Time (America/New York)
Monday	16:00:00 to 20:00:00
Tuesday	16:00:00 to 20:00:00
Wednesday	16:00:00 to 20:00:00
Thursday	16:00:00 to 20:00:00
Friday	16:00:00 to 20:00:00

Holidays

LEAN uses the trading holidays from the NYSE website.

The following table shows the dates of holidays for the US Equity market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01

Date (*yyyy-mm-dd*)				
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01

Date (*yyyy-mm-dd*)				
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	

Early Closes

The following table shows the early closes for the US Equity market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
1999-11-26	13:00:00
2000-07-03	13:00:00
2000-11-24	13:00:00
2001-07-03	13:00:00
2001-11-23	13:00:00
2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-07-03	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2018-07-03	13:00:00
2018-11-23	13:00:00
2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The US Equity market trades in the `America/New York` time zone.

Asset Classes

India Equity

India Equities represent partial ownership in an Indian corporation.

Requesting Data

Handling Data

Corporate Actions

Data Preparation

Market Hours

See Also

BasicTemplateIndiaAlgorithm.py
BasicTemplateIndiaAlgorithm.cs

India Equity

Requesting Data

Introduction

Request India Equity data in your algorithm to receive a feed of asset prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the India Equities dataset listing . To trade India Equities live, you can use one of the brokerage data feeds .

Create Subscriptions

To create an India Equity subscription, in the `Initialize` method, call the `AddEquity` method. The `AddEquity` method returns an `Equity` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
PY
self.symbol = self.AddEquity("YESBANK", market=Market.India).Symbol
```

If you set the brokerage model to an India brokerage, you don't need to pass a `market` argument. To view the integrated brokerages that offer India Equities, see Brokerages .

To view the supported assets in the India Equities dataset, see the Data Explorer .

Resolutions

The following table shows the available resolutions and data formats for India Equity subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓			
`Hour`	✓			
`Daily`	✓			

The default resolution for India Equity subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddEquity` method.

```
PY
self.symbol = self.AddEquity("YESBANK", Resolution.Daily, Market.India).Symbol
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

LEAN groups all of the India Equity exchanges under `Market.India` . To set the market for a security, pass a `market` argument to the `AddEquity` method.

```
self.symbol = self.AddEquity("YESBANK", market=Market.India).Symbol
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.symbol = self.AddEquity("YESBANK", market=Market.India, fillForward=False).Symbol
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The amount of margin that's available depends on the brokerage model you use. For more information about the margin requirements of each brokerage, see the **Margin** section of the brokerage guides . To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddEquity` method.

```
self.symbol = self.AddEquity("YESBANK", market=Market.India, leverage=3).Symbol
```

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.symbol = self.AddEquity("YESBANK", market=Market.India, extendedMarketHours=True).Symbol
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Data Normalization

The data normalization mode defines how historical data is adjusted for corporate actions . The data normalization mode affects the data that LEAN passes to `OnData` and the data from history request . By default, LEAN adjusts India Equity data for splits and dividends to produce a smooth price curve. The following data normalization modes are

available:

We use the entire split and dividend history to adjust historical prices. This process ensures you get the same adjusted prices, regardless of the backtest end date.

To set the data normalization mode for a security, pass a `dataNormalizationMode` argument to the `AddEquity` method..

```
self.symbol = AddEquity("YESBANK", market=Market.India,
dataNormalizationMode=DataNormalizationMode.Raw).Symbol
```

PY

Properties

The `AddEquity` method returns an `Equity` object, which have the following properties:

India Equity

Handling Data

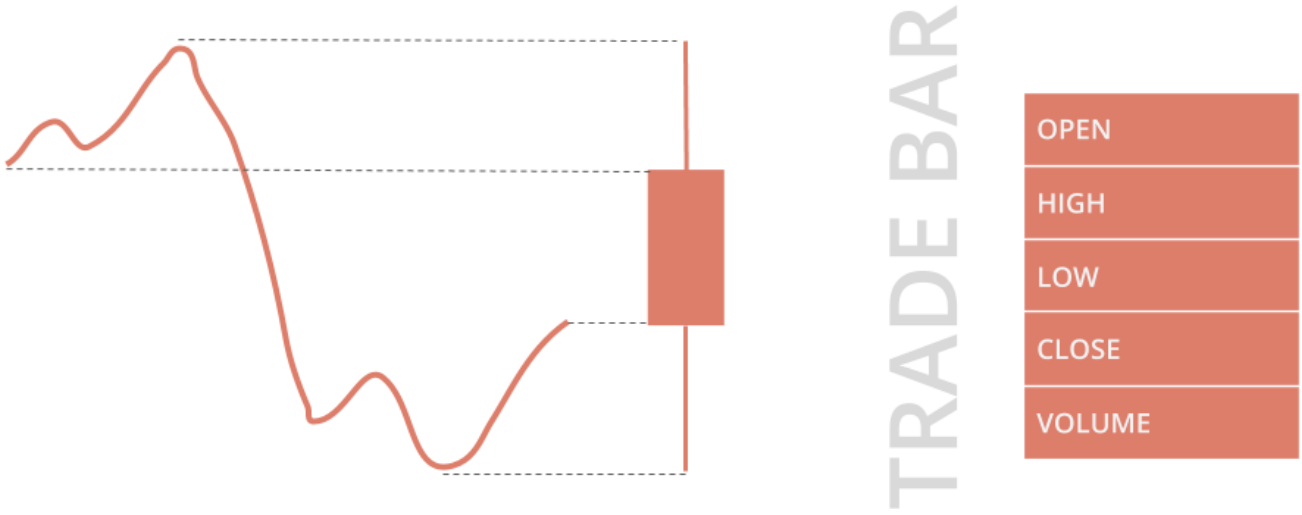
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

India Equity

Corporate Actions

Introduction

India Equity subscriptions provide notifications for splits, dividends, symbol changes, and delistings.

Splits

When a company does a stock split, the number of shares each shareholder owns increases and the price of each share decreases. When a company does a reverse stock split, the number of shares each shareholder owns decreases and the price of each share increases. A company may perform a stock split or a reverse stock split to adjust the price of their stock so that more investors trade it and the liquidity increases.

When a stock split or reverse stock split occurs for an Equity in your algorithm, LEAN sends a `Split` object to the `OnData` method. Split objects have the following properties:

You receive `Split` objects when a split is in the near future and when it occurs. To know if the split occurs in the near future or now, check the `Type` property.

If you backtest without the `Raw` data normalization mode , the splits are factored into the price and volume. If you backtest with the `Raw` data normalization mode or trade live, when a split occurs, LEAN automatically adjusts your positions based on the `SplitFactor` . If the post-split quantity isn't a valid lot size , LEAN credits the remaining value to your cashbook in your account currency. If you have indicators in your algorithm, reset your indicators when splits occur so that the data in your indicators account for the price adjustments that the splits cause.

To get the `Split` objects in the `Slice` , index the `Splits` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Splits` property contains data for your security before you index it with the security `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    split = slice.Splits.get(self.symbol)
    if split:
        pass
```

You can also iterate through the `Splits` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Split` objects.

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, split in slice.Splits.items():
        pass
```

LEAN stores the data for stock splits in factor files. To view some example factor files, see the LEAN GitHub repository . In backtests, your algorithm receives `Split` objects at midnight. In live trading, your algorithm receives `Split` objects

when the factor files are ready.

If you hold an Option contract for an underlying Equity when a split occurs, LEAN closes your Option contract position.

If a split event occurs before your order is filled, the unfilled portion of the order is adjusted automatically, where its quantity is multiplied by the split factor and the limit/stop/trigger price (if any) is divided by the split factor.

Dividends

A dividend is a payment that a company gives to shareholders to distribute profits. When a dividend payment occurs for an Equity in your algorithm, LEAN sends a `Dividend` object to the `OnData` method. `Dividend` objects have the following properties:

If you backtest with the `Adjusted` or `TotalReturn` data normalization mode, the dividends are factored into the price. If you backtest with the other data normalization modes or trade live, when a dividend payment occurs, LEAN automatically adds the payment amount to your cashbook. If you have indicators in your algorithm, reset your indicators when dividend payments occur so that the data in your indicators account for the price adjustments that the dividend causes.

To get the `Dividend` objects in the `Slice` , index the `Dividends` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Dividends` property contains data for your security before you index it with the security `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    dividend = slice.Dividends.get(self.symbol)
    if dividend:
        pass
```

You can also iterate through the `Dividends` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Dividend` objects.

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, dividend in slice.Dividends.items():
        pass
```

For a full example, see the DividendAlgorithm in the LEAN GitHub repository.

Symbol Changes

The benefit of the `Symbol` class is that it always maps to the same security, regardless of their trading ticker. When a company changes its trading ticker, LEAN sends a `SymbolChangedEvent` to the `OnData` method. `SymbolChangedEvent` objects have the following properties:

To get the `SymbolChangedEvent` objects in the `Slice` , index the `SymbolChangedEvents` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `SymbolChangedEvents` property contains data for your security before you index it with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    symbol_changed_event = slice.SymbolChangedEvents.get(self.symbol)
    if symbol_changed_event:
        pass
```

You can also iterate through the `SymbolChangedEvents` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `SymbolChangedEvent` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, symbol_changed_event in slice.SymbolChangedEvents.items():
        pass
```

If you have an open order for a security when they change their ticker, LEAN cancels your order. To keep your order, in the `OnOrderEvent` method, get the quantity and `Symbol` of the cancelled order and submit a new order.

```
def OnOrderEvent(self, order_event: OrderEvent) -> None:
    if order_event.Status == OrderStatus.Canceled:
        ticket = self.Transactions.GetOrderTicket(order_event.OrderId)
        if "symbol changed event" in ticket.Tag:
            self.Transactions.AddOrder(ticket.SubmitRequest)
```

LEAN stores the data for ticker changes in map files. To view some example map files, see the LEAN GitHub repository .

Delistings

When a company is delisting from an exchange, LEAN sends a `Delisting` object to the `OnData` method. `Delisting` objects have the following properties:

You receive `Delisting` objects when a delisting is in the near future and when it occurs. To know if the delisting occurs in the near future or now, check the `Type` property.

To get the `Delisting` objects in the `Slice` , index the `Delistings` property of the `Slice` with the security `Symbol` . The `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Delistings` property contains data for your security before you index it with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    delisting = slice.Delistings.get(self.symbol)
    if delisting:
        pass
```

You can also iterate through the `Delistings` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Delisting` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, delisting in slice.Delistings.items():
        pass
```

The delist warning occurs on the final trading day of the stock to give you time to gracefully exit out of positions before LEAN automatically liquidates them.

```py
if delisting.Type == DelistingType.Warning:
    # Liquidate with MarketOnOpenOrder on delisting warning
    quantity = self.Portfolio[symbol].Quantity
    if quantity != 0:
        self.MarketOnOpenOrder(symbol, -quantity)
```

For a full example, see the DelistingEventsAlgorithm in the LEAN GitHub repository.

India Equity

Data Preparation

Introduction

The India Equities dataset provides price data for backtests.

Sourcing

TrueData is an authorized NSE and MCX data vendor founded by Kapil Marwaha in 2007, with the goal of providing users with a multitude of solutions for the financial services sector, including making market data feeds available in all its forms, to as many applications as possible, including a wide range of technical analysis applications, trading solutions, automated & algorithmic traders/platforms and a lot more.

Bar Building

We receive minute-resolution trade data and aggregate it to build hour and daily bars. To create custom resolution periods, see Consolidating Data .

India Equity

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the India Equity market.

Pre-market Hours

The following table shows the pre-market hours for the India Equity market:

Weekday	Time (Asia/Kolkata)
Monday	09:00:00 to 09:15:00
Tuesday	09:00:00 to 09:15:00
Wednesday	09:00:00 to 09:15:00
Thursday	09:00:00 to 09:15:00
Friday	09:00:00 to 09:15:00

Regular Trading Hours

The following table shows the regular trading hours for the India Equity market:

Weekday	Time (Asia/Kolkata)
Monday	09:15:00 to 15:30:00
Tuesday	09:15:00 to 15:30:00
Wednesday	09:15:00 to 15:30:00
Thursday	09:15:00 to 15:30:00
Friday	09:15:00 to 15:30:00

Post-market Hours

The following table shows the post-market hours for the India Equity market:

Weekday	Time (Asia/Kolkata)
Monday	15:40:00 to 16:00:00
Tuesday	15:40:00 to 16:00:00
Wednesday	15:40:00 to 16:00:00
Thursday	15:40:00 to 16:00:00
Friday	15:40:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the India Equity market:

Date (*yyyy-mm-dd*)				
2004-01-26	2004-04-14	2005-01-26	2005-04-14	2005-08-15
2006-01-26	2006-04-14	2006-05-01	2006-08-15	2006-10-02
2006-12-25	2007-01-26	2007-05-01	2007-08-15	2007-10-02
2007-12-25	2008-04-14	2008-05-01	2008-08-15	2008-10-02
2008-12-25	2009-01-26	2009-04-14	2009-05-01	2009-10-02
2009-12-25	2010-01-26	2010-04-14	2011-01-26	2011-03-02
2011-04-12	2011-04-14	2011-04-22	2011-08-15	2011-08-31
2011-09-01	2011-10-06	2011-10-26	2011-10-27	2011-11-07
2011-11-10	2011-12-06	2012-01-26	2012-02-20	2012-03-08
2012-04-05	2012-04-06	2012-05-01	2012-08-15	2012-08-20
2012-09-19	2012-10-02	2012-10-24	2012-11-14	2012-11-28
2012-12-25	2013-03-27	2013-03-29	2013-04-19	2013-04-24
2013-05-01	2013-08-09	2013-08-15	2013-09-09	2013-10-02
2013-10-16	2013-11-04	2013-11-15	2013-12-25	2014-02-27
2014-03-17	2014-04-08	2014-04-14	2014-04-18	2014-04-24
2014-05-01	2014-07-29	2014-08-15	2014-08-29	2014-10-02
2014-10-03	2014-10-06	2014-10-15	2014-10-24	2014-11-04
2014-11-06	2014-12-25	2015-01-26	2015-02-17	2015-03-06

Date (*yyyy-mm-dd*)				
2015-04-02	2015-04-03	2015-04-14	2015-05-01	2015-09-17
2015-09-25	2015-10-02	2015-10-22	2015-11-12	2015-11-25
2015-12-25	2016-01-26	2016-03-07	2016-03-24	2016-03-25
2016-04-14	2016-04-15	2016-04-19	2016-07-06	2016-08-15
2016-09-05	2016-09-13	2016-10-11	2016-10-12	2016-10-31
2016-11-14	2017-01-26	2017-02-24	2017-03-13	2017-04-04
2017-04-14	2017-05-01	2017-06-26	2017-08-15	2017-08-25
2017-10-02	2017-10-20	2017-12-25	2018-01-26	2018-02-13
2018-03-02	2018-03-29	2018-03-30	2018-05-01	2018-08-15
2018-08-22	2018-09-13	2018-09-20	2018-10-02	2018-10-18
2018-11-08	2018-11-23	2018-12-25	2019-03-04	2019-03-21
2019-04-17	2019-04-19	2019-04-29	2019-05-01	2019-06-05
2019-08-12	2019-08-15	2019-09-02	2019-09-10	2019-10-02
2019-10-08	2019-10-21	2019-10-28	2019-11-12	2019-12-25
2020-02-21	2020-03-10	2020-04-02	2020-04-06	2020-04-10
2020-04-14	2020-05-01	2020-05-25	2020-10-02	2020-11-16
2020-11-30	2020-12-25	2021-01-26	2021-03-11	2021-03-29
2021-04-02	2021-04-14	2021-04-21	2021-05-13	2021-07-21
2021-08-19	2021-09-10	2021-10-15	2021-11-05	2021-11-19
2022-01-26	2022-04-14	2022-08-15	2023-01-26	2023-04-14
2023-05-01	2023-08-15	2023-10-02	2023-12-25	2024-01-26
2024-05-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The India Equity market trades in the `Asia/Kolkata` time zone.

Asset Classes

Equity Options

Equity Options are a financial derivative that gives the holder the right (but not the obligation) to buy or sell the underlying Equity, such as Apple, at the stated exercise price.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateOptionsAlgorithm.py
BasicTemplateOptionsAlgorithm.cs

Equity Options

Requesting Data

Introduction

Request Equity Options data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Equity Options dataset listing . To trade Equity Options live, you can use one of the brokerage data feeds . We currently only support American-style Options for US Equity Options.

Create Subscriptions

Before you can subscribe to an Option contract, you must configure the underlying Equity and get the contract `Symbol` .

Configure the Underlying Equity

If you want to subscribe to the underlying Equity in the `Initialize` method, set the Equity data normalization to `DataNormalizationMode.Raw` .

```
self.symbol = self.AddEquity("SPY", dataNormalizationMode=DataNormalizationMode.Raw).Symbol
```

If your algorithm has a dynamic universe of Equities, before you add the Equity universe in the `Initialize` method, set the universe data normalization mode to `DataNormalizationMode.Raw` .

```
self.UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw
```

If you subscribe to an Equity Option contract but don't have a subscription to the underlying Equity, LEAN automatically subscribes to the underlying Equity with the following settings:

Setting	Value
Fill forward	Same as the Option contract
Leverage	0
Extended Market Hours	Same as the Option contract
Data Normalization	`DataNormalizationMode.Raw`

In this case, you still need the Equity `Symbol` to subscribe to Equity Option contracts. If you don't have access to it, create it.

```
self.symbol = Symbol.Create("SPY", SecurityType.Equity, Market.USA)
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Get Contract Symbols

To subscribe to an Option contract, you need the contract `Symbol` . You can get the contract `Symbol` from the `CreateOption` method or from the `OptionChainProvider` . If you use the `CreateOption` method, you need to provide the details of an existing contract.

```
self.contract_symbol = Symbol.CreateOption(self.symbol, Market.USA,
    OptionStyle.American, OptionRight.Call, 365, datetime(2022, 6, 17))
```

Another way to get an Option contract `Symbol` is to use the `OptionChainProvider` . The `GetOptionContractList` method of `OptionChainProvider` returns a list of `Symbol` objects that reference the available Option contracts for a given underlying Equity on a given date. To filter and select contracts, you can use the following properties of each `Symbol` object:

Property	Description
`ID.Date`	The expiration date of the contract.
`ID.StrikePrice`	The strike price of the contract.
`ID.OptionRight`	The contract type. The `OptionRight` enumeration has the following members:
`ID.OptionStyle`	The contract style. The `OptionStyle` enumeration has the following members: We currently only support American-style Options for US Equity Options.

```
contract_symbols = self.OptionChainProvider.GetOptionContractList(self.symbol, self.Time)
expiry = min([symbol.ID.Date for symbol in contract_symbols])
filtered_symbols = [symbol for symbol in contract_symbols if symbol.ID.Date == expiry and
symbol.ID.OptionRight == OptionRight.Call]
self.contract_symbol = sorted(filtered_symbols, key=lambda symbol: symbol.ID.StrikePrice)[0]
```

Subscribe to Contracts

To create an Equity Option contract subscription, pass the contract `Symbol` to the `AddOptionContract` method. Save a reference to the contract `Symbol` so you can easily access the Option contract in the OptionChain that LEAN passes to the `OnData` method. This method returns an `Option` object. To override the default pricing model of the Option, set a pricing model .

```
option = self.AddOptionContract(self.contract_symbol)
option.PriceModel = OptionPriceModels.BjerksundStensland()
```

The `AddOptionContract` method creates a subscription for a single Option contract and adds it to your **user-defined** universe. To create a dynamic universe of Option contracts, add an Equity Options universe or an Options Universe Selection model .

Warm Up Contract Prices

If you subscribe to an Option contract with `AddOptionContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
PY
seeder = FuncSecuritySeeder(self.GetLastKnownPrices)
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel, seeder, self))
```

Supported Assets

To view the supported assets in the US Equities dataset, see the Data Explorer .

Resolutions

The following table shows the available resolutions and data formats for Equity Option contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Option contract subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddOptionContract` method.

```
PY
self.AddOptionContract(self.contract_symbol, Resolution.Minute)
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

LEAN groups all of the US Equity exchanges under `Market.USA` . You don't need to pass a `Market` argument to the `AddOptionContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.AddOptionContract(self.contract_symbol, fillForward=False)
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Options are already leveraged products, so you can't change their leverage.

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.AddOptionContract(self.contract_symbol, extendedMarketHours=True)
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Remove Subscriptions

To remove a contract subscription that you created with `AddOptionContract` , call the `RemoveOptionContract` method. This method is an alias for `RemoveSecurity` .

```
self.RemoveOptionContract(self.contract_symbol)
```

The `RemoveOptionContract` method cancels your open orders for the contract and liquidates your holdings.

Properties

The `AddOptionContract` method returns an `Option` object, which have the following properties:

Helper Methods

The `Option` object provides methods you can use for basic calculations. These methods require the underlying price. To get the `Option` object and the `Security` object for its underlying in any function, use the Option `Symbol` to access the value in the `Securities` object.

```
option = self.Securities[self.contract_symbol]
underlying = self.Securities[self.contract_symbol.Underlying]
underlying_price = underlying.Price
```

To get the Option payoff , call the `GetPayOff` method.

```py
pay_off = option.GetPayOff(underlying_price)
```

To get the Option intrinsic value , call the `GetIntrinsicValue` method.

```py
intrinsic_value = option.GetIntrinsicValue(underlying_price)
```

To get the Option out-of-the-money amount , call the `OutOfTheMoneyAmount` method.

```py
otm_amount = option.OutOfTheMoneyAmount(underlying_price)
```

To check whether the Option can be automatic exercised, call the `IsAutoExercised` method.

```py
is_auto_exercised = option.IsAutoExercised(underlying_price)
```

Equity Options

Handling Data

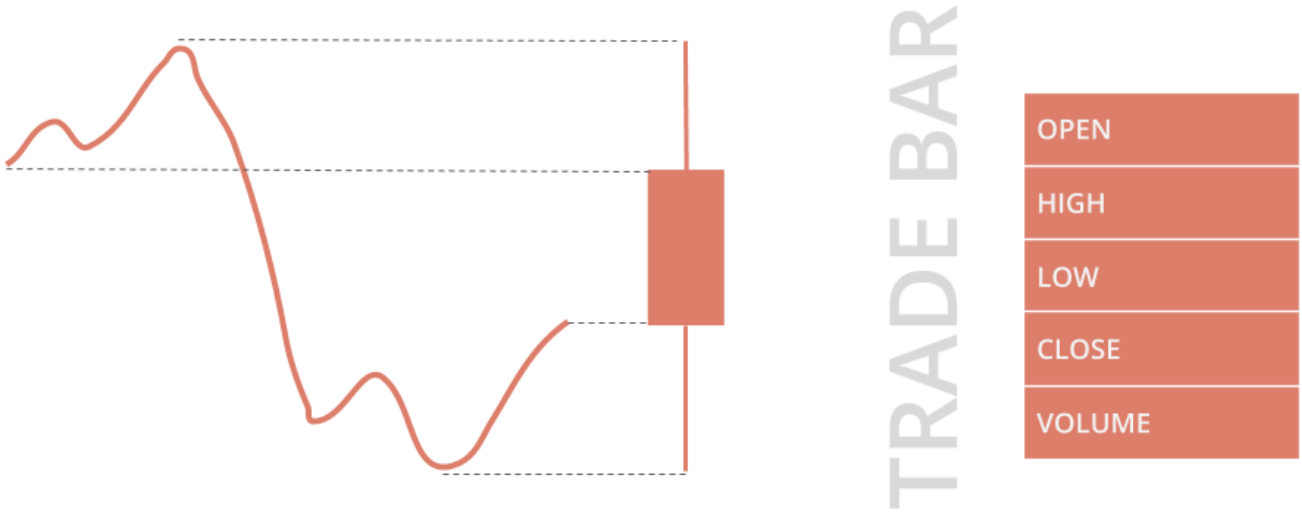
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the contract ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

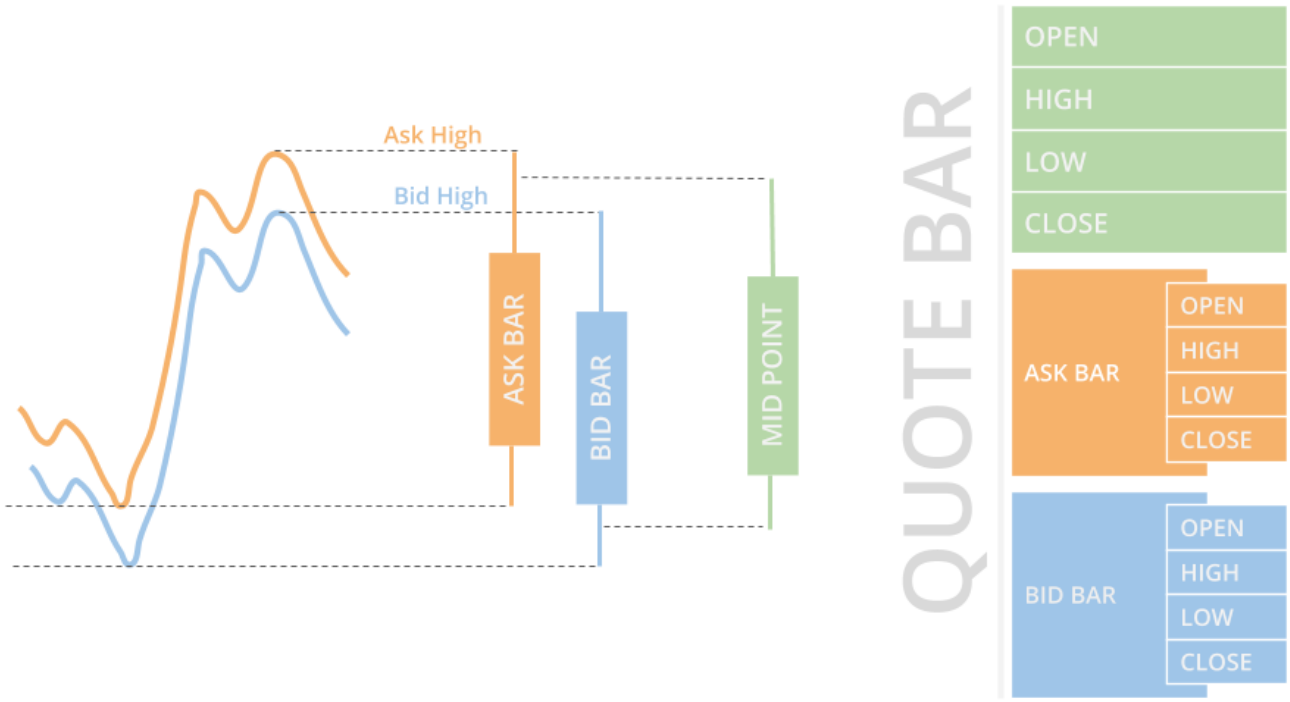
```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.Bars:
        trade_bar = slice.Bars[self.contract_symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively get quotes or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.contract_symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security. They have the following properties:

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contracts = chain.Contracts
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.OptionChains.items():
        contracts = chain.Contracts
```

Option Contracts

`OptionContract` objects represent the data of a single Option contract in the market. They have the following properties:

To get the Option contracts in the `Slice` , use the `Contracts` property of the `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            price = contract.Price
```

Greeks and Implied Volatility

To get the Greeks and implied volatility of an Option contract, use the `Greeks` and `ImpliedVolatility` members.

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            delta = contract.Greeks.Delta
            iv = contract.ImpliedVolatility
```

LEAN only calculates Greeks and implied volatility when you request them because they are expensive operations. If you invoke the `Greeks` member, the Greeks aren't calculated. However, if you invoke the `Greeks.Delta` member, LEAN calculates the delta. To avoid unecessary computation in your algorithm, only request the Greeks and implied volatility when you need them. For more information about the Greeks and implied volatility, see Options Pricing .

Open Interest

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Option` or `OptionContract` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            open_interest = contract.OpenInterest
```

Equity Options

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the Equity Option market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Equity Option market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

LEAN uses the trading holidays from the NYSE website.

The following table shows the dates of holidays for the Equity Option market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	

Early Closes

The following table shows the early closes for the Equity Option market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
1999-11-26	13:00:00
2000-07-03	13:00:00
2000-11-24	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2001-07-03	13:00:00
2001-11-23	13:00:00
2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00
2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2012-12-24	13:00:00
2013-07-03	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00
2018-07-03	13:00:00
2018-11-23	13:00:00
2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The Equity Option market trades in the `America/New York` time zone.

Asset Classes

Crypto

A Cryptocurrency is a digital currency that's secured by cryptography, which makes it nearly impossible to counterfeit or double-spend.

Requesting Data

Handling Data

Holdings

Market Hours

See Also

BasicTemplateCryptoAlgorithm.py
BasicTemplateCryptoAlgorithm.cs

Crypto

Requesting Data

Introduction

Request Crypto data in your algorithm to receive a feed of asset prices in the `OnData` method. For more information about the specific datasets we use for backtests, see the CoinAPI datasets . To trade Crypto live, you can use our Crypto data feed .

Create Subscriptions

To create a Crypto subscription, in the `Initialize` method, call the `AddCrypto` method. The `AddCrypto` method returns a `Crypto` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
self.symbol = self.AddCrypto("BTCUSD").Symbol
```

The `AddCrypto` method creates a subscription for a single Crypto asset and adds it to your **user-defined** universe. To create a dynamic universe of Crypto assets, add a Crypto universe .

To view the supported assets, see the CoinAPI datasets .

Fungible Assets

Fungible assets are assets that are interchangeable with each other. For example, every Bitcoin is effectively identical to every other Bitcoin. In contrast, non-fungible tokens (NFTs) are non-fungible because each one is unique. An example of an NFT is the Bored Ape Yacht Club collection. LEAN doesn't currently support trading NFTs.

Resolutions

The following table shows the available resolutions and data formats for Crypto subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Crypto subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution`

argument to the `AddCrypto` method.

```
self.symbol = self.AddCrypto("BTCUSD", Resolution.Daily).Symbol
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Crypto:

To set the market for a security, pass a `market` argument to the `AddCrypto` method.

```
self.symbol = self.AddCrypto("BTCUSD", market=Market.GDAX).Symbol
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.symbol = self.AddCrypto("BTCUSD", fillForward=False).Symbol
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The amount of leverage available to you depends on the brokerage you use. To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddCrypto` method.

```
self.symbol = self.AddCrypto("BTCUSD", leverage=3).Symbol
```

For more information about the leverage each brokerage provides, see Brokerages .

Properties

The `AddCrypto` method returns a `Crypto` object, which have the following properties:

Crypto

Handling Data

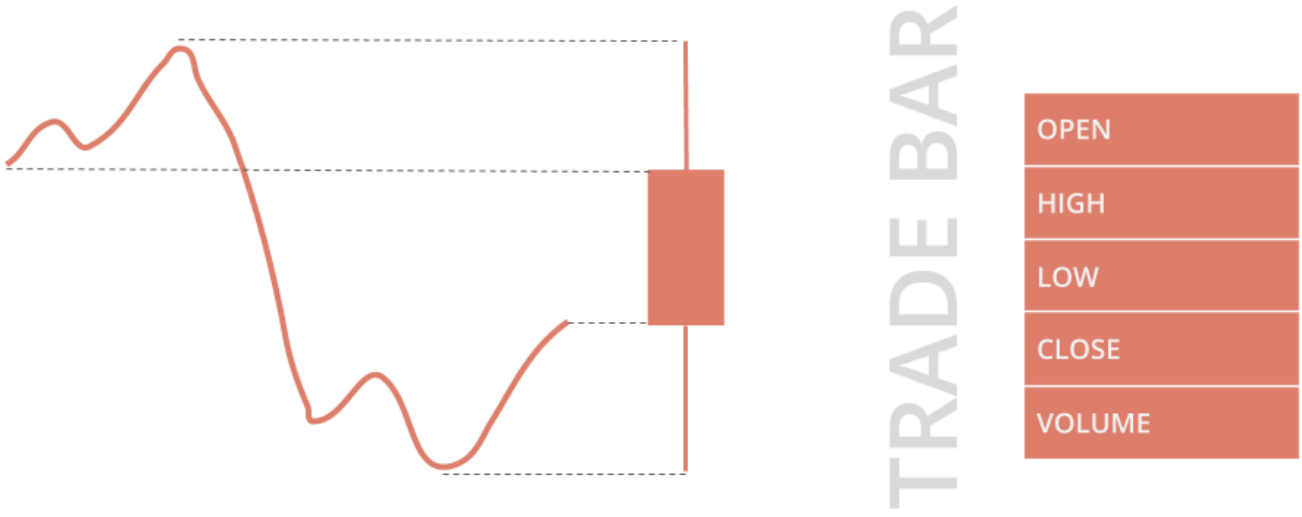
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

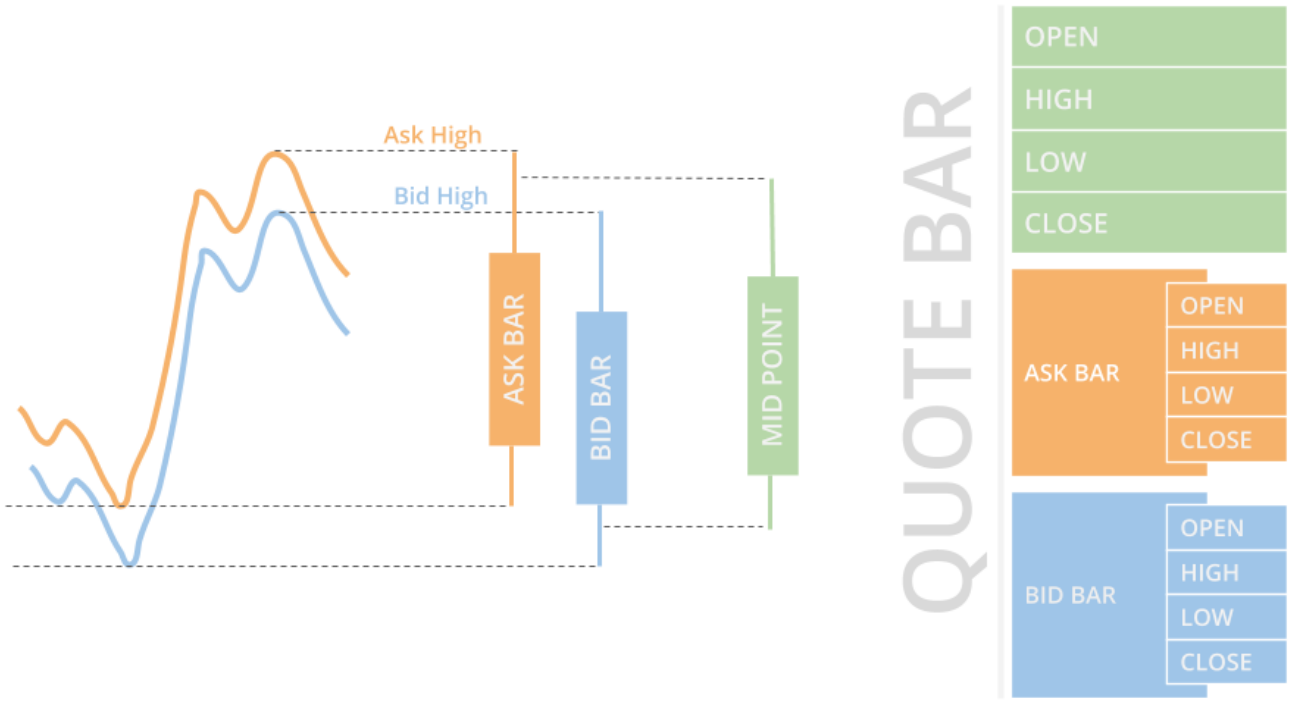
```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the security `Symbol` . If the security doesn't actively get quotes or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the security. A quote tick is an offer to buy or sell the security at a specific price. `Tick` objects have the following properties:

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            price = tick.Price
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `list[Tick]` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            price = tick.Price
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able

to perform your own online tick filtering.

Crypto

Holdings

Introduction

Crypto holdings require special consideration because not all brokerages track your positions in each currency pair.

Cash vs Margin Brokerage Accounts

Some of the Crypto brokerages integrated into QuantConnect support cash and margin accounts while some only support cash accounts. If you want to short Crypto assets or use leverage, you need a margin account. Follow these steps to view which account types each brokerage supports:

1. Open the brokerage guides .
2. Click a Crypto brokerage.
3. Scroll down to the **Account Types** section.

Virtual Pairs

All fiat and Crypto currencies are individual assets. When you buy a pair like BTCUSD, you trade USD for BTC. In this case, LEAN removes some USD from your portfolio cashbook and adds some BTC. The virtual pair BTCUSD represents your position in that trade, but the virtual pair doesn't actually exist. It simply represents an open trade. If you call the Liquidate method with the `Symbol` of a virtual pair, the method only liquidates the quantity in your virtual pair. For more information about liquidating Crypto postitions, see Crypto Trades .

Access Cypto Holdings

The `CashBook` stores the quantity of your Crypto holdings. To access your Crypto holdings, index the `CashBook` with the currency ticker. The values of the `CashBook` dictionary are `Cash` objects, which have the following properties:

```
PY
btc_quantity = self.Portfolio.CashBook['BTC'].Amount
btc_value = self.Portfolio.CashBook['BTC'].ValueInAccountCurrency
```

To access the virtual pair of your Crypto trades, index the `Portfolio` object with the pair `Symbol` .

```
PY
security_holding  = self.Portfolio[self.btc_usd_symbol]
```

Stateful Redeployments

When you make a trade inside of a running algorithm, LEAN tracks the virtual position state for you, but it won't survive between deployments. Some brokerages save your virtual pairs, so you can load them into your algorithm when you stop and redeploy it. Depending on the brokerage you select, you may be able to manually add virtual pairs when you deploy live algorithms with the LEAN CLI or cloud deployment wizard .

Crypto

Market Hours

Introduction

The Crypto markets are open 24/7. There are no pre-market hours, post-market hours, holidays, late opens, or early closes.

Asset Classes

Crypto Futures

Crypto Perpetual Futures are derivative financial contracts that obligate parties to buy or sell an asset at an unspecified future date.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateCryptoFutureAlgorithm.py
BasicTemplateCryptoFutureAlgorithm.cs

Crypto Futures

Requesting Data

Introduction

Request Crypto Futures data in your algorithm to receive a feed of asset prices in the `OnData` method. For more information about the specific datasets we use for backtests, see the Binance Crypto Future Price Data dataset listing . To trade Crypto live, you can use our Crypto Futures data feed .

Create Subscriptions

To create a Crypto Futures subscription, in the `Initialize` method, call the `AddCryptoFuture` method. The `AddCryptoFuture` method returns a `CryptoFuture` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
self.symbol = self.AddCryptoFuture("BTCUSD").Symbol
```

The `AddCryptoFuture` method creates a subscription for a single Crypto Futures asset and adds it to your **user-defined** universe.

To view the supported assets in the Crypto Futures dataset, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Crypto Futures contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Crypto Futures subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddCryptoFuture` method.

```
self.symbol = self.AddCryptoFuture("BTCUSD", Resolution.Daily).Symbol
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

Crypto Futures are currently only available on `Market.Binance` . To set the market for a security, pass a `market` argument to the `AddCryptoFuture` method.

```
self.symbol = self.AddCryptoFuture("BTCUSD", market=Market.Binance).Symbol
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.symbol = self.AddCryptoFuture("BTCUSD", fillForward=False).Symbol
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The amount of leverage available to you depends on the brokerage you use. To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddCryptoFuture` method.

```
self.symbol = self.AddCryptoFuture("BTCUSD", leverage=3).Symbol
```

For more information about the leverage each brokerage provides, see Brokerages .

Properties

The `AddCryptoFuture` method returns a `CryptoFuture` object, which have the following properties:

Crypto Futures

Handling Data

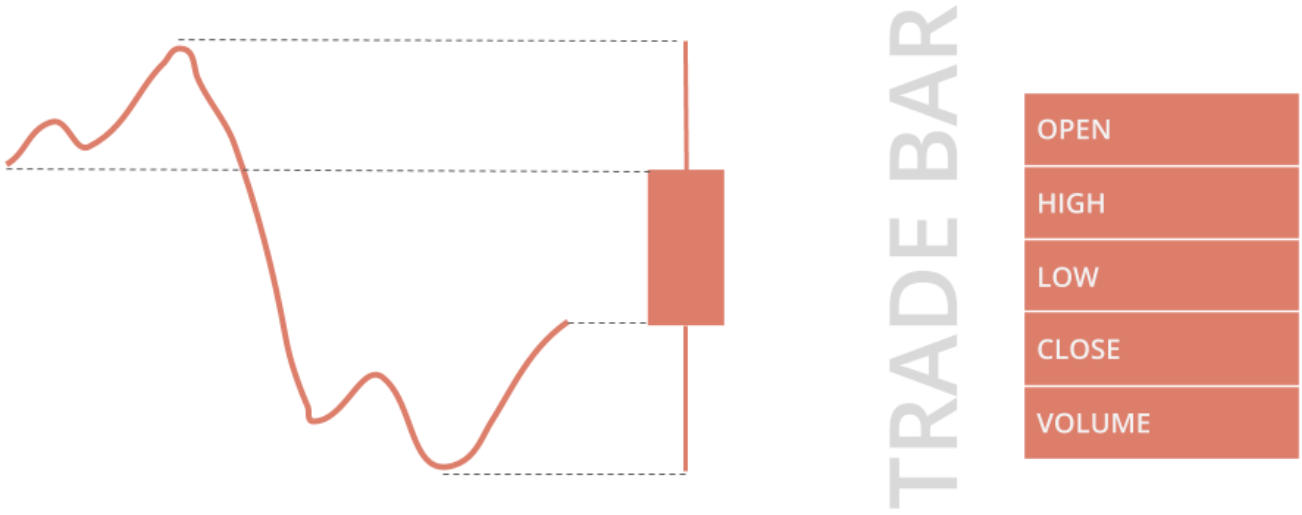
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

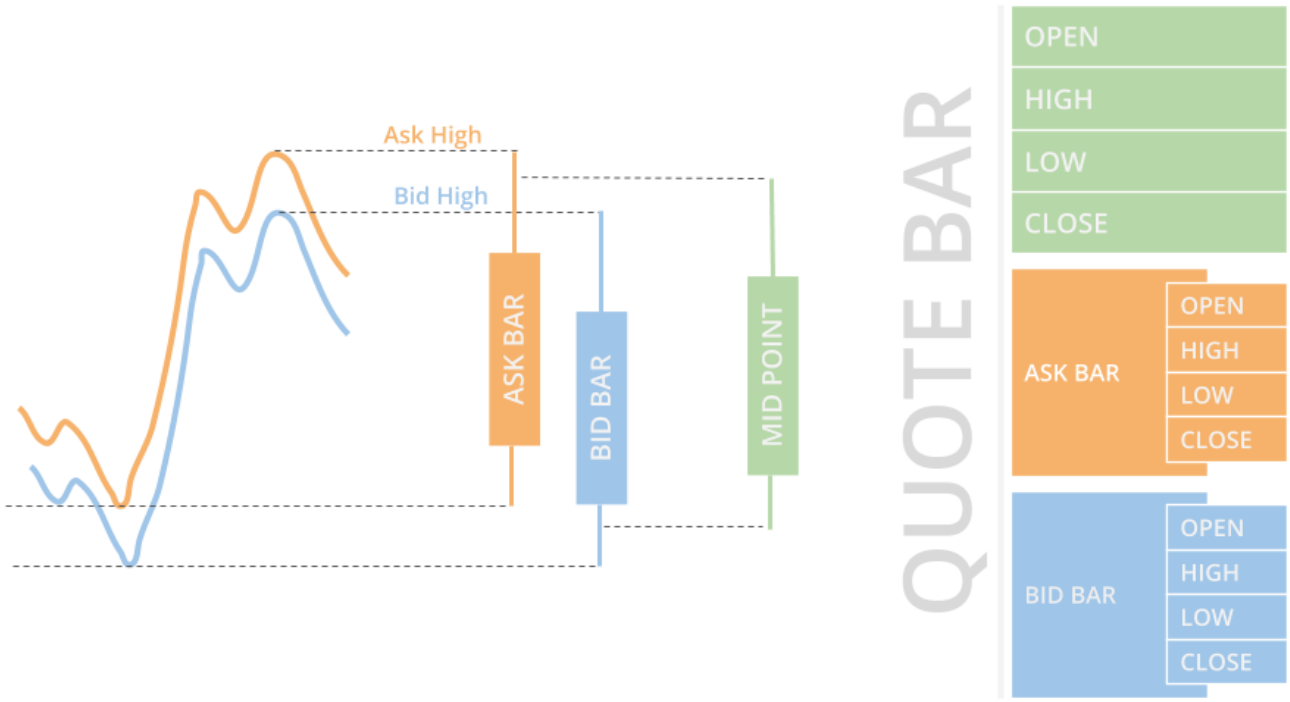
```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the security `Symbol` . If the security doesn't actively get quotes or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the security. A quote tick is an offer to buy or sell the security at a specific price. `Tick` objects have the following properties:

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            price = tick.Price
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `list[Tick]` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            price = tick.Price
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able

to perform your own online tick filtering.

Margin Interest Rates

`MarginInterestRate` objects contain the margin interest rate, which is a cost associated with trading on margin. `MarginInterestRate` objects have the following properties:

To get the `MarginInterestRate` objects in the `Slice` , index the `MarginInterestRate` property of the `Slice` with the Crypto Future `Symbol` . The `MarginInterestRate` property of the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the property contains data for your Crypto Future before you index it with the Crypto Future `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.MarginInterestRates:
        interest_rate = slice.MarginInterestRates[self.symbol].InterestRate
```

You can also iterate through the `MarginInterestRates` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `MarginInterestRate` objects.

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, margin_interest_rate in slice.MarginInterestRates.items():
        interest_rate = margin_interest_rate.InterestRate
```

Crypto Futures

Market Hours

Introduction

The Crypto Futures markets are open 24/7. There are no pre-market hours, post-market hours, holidays, late opens, or early closes.

Asset Classes

Forex

Forex markets are for trading foreign currencies.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateForexAlgorithm.py
BasicTemplateForexAlgorithm.cs

Forex

Requesting Data

Introduction

Request Forex data in your algorithm to receive a feed of exchange rates in the `OnData` method. For more information about the specific dataset we use for backtests, see the FOREX dataset listing . To trade Forex live, you can use our Forex data feed .

Create Subscriptions

To create a Forex pair subscription, in the `Initialize` method, call the `AddForex` method. The `AddForex` method returns a `Forex` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
self.symbol = self.AddForex("EURUSD").Symbol
```

To view the supported Forex pairs, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Forex subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				✓
`Second`		✓		
`Minute`		✓		
`Hour`		✓		
`Daily`		✓		

The default resolution for Forex subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddForex` method.

```
self.symbol = self.AddForex("EURUSD", Resolution.Daily).Symbol
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The only market available for Forex pairs is `Market.Oanda` , so you don't need to pass a `market` argument to the `AddForex` method.

```
self.symbol = self.AddForex("EURUSD", market=Market.Oanda).Symbol
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.symbol = self.AddForex("EURUSD", fillForward=False).Symbol
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The Oanda brokerage let's you use up to 50x leverage on margin accounts. To change the amount of leverage you can use for a Forex pair, pass a `leverage` argument to the `AddForex` method.

```
self.symbol = self.AddForex("EURUSD", leverage=35).Symbol
```

Properties

The `AddForex` method returns a `Forex` object, which have the following properties:

Forex

Handling Data

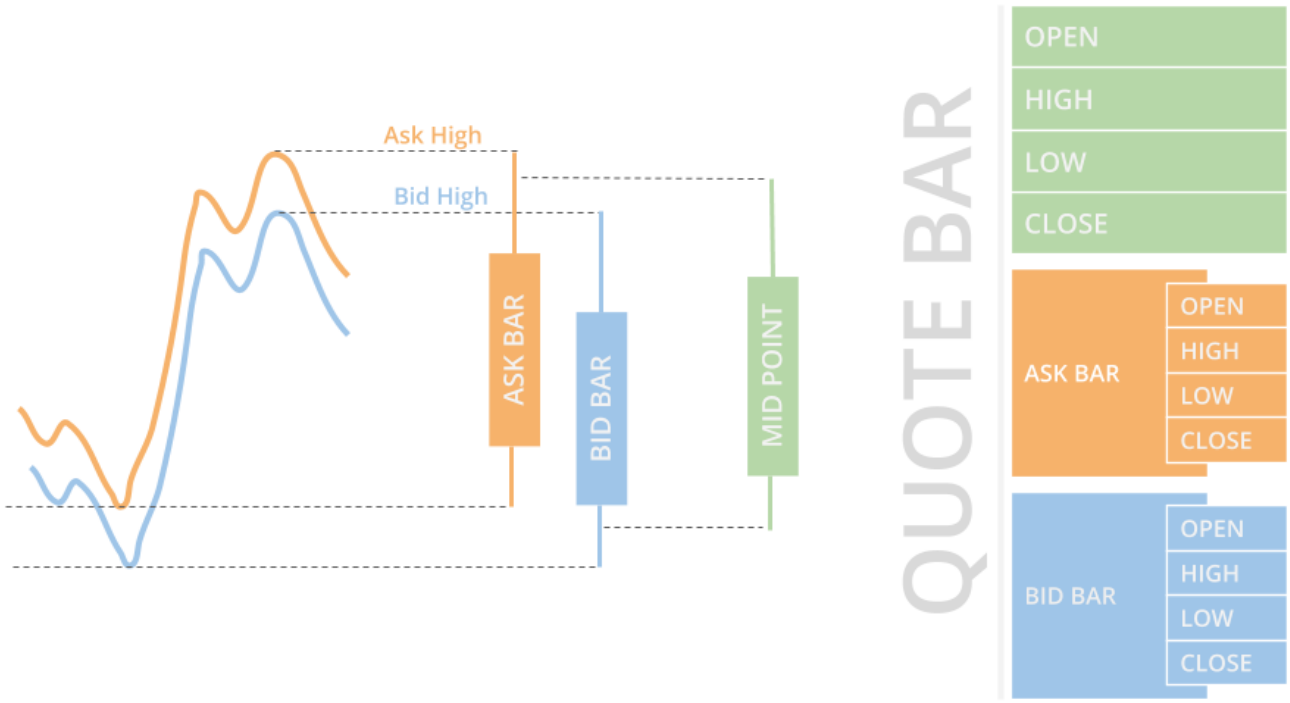
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `QuoteBars DataDictionary` is made up of `QuoteBar` objects. To access individual data points in the dictionary, you can index the dictionary with the Forex pair ticker or `Symbol` , but we recommend you use the `Symbol` .

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the Forex pair `Symbol` . If the Forex pair doesn't actively get quotes or you are in the same time step as when you added the Forex pair subscription,

the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Forex pair before you index the `Slice` with the Forex pair `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the Forex pair. A quote tick is an offer to buy or sell the Forex pair at a specific price. `Tick` objects have the following properties:

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the Forex pair doesn't actively trade or you are in the same time step as when you added the Forex pair subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Forex pair before you index the `Slice` with the Forex pair `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            price = tick.Price
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `list[Tick]` objects.

```py
def OnData(self, slice: Slice) -> None:
    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            price = tick.Price
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

Forex

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Forex market:

Weekday	Time (America/New York)
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The Forex market trades in the `America/New York` time zone.

Assets With Other Hours

The following list shows the pairs that have different trading periods than the overall Forex market:

- EURNZD
- GBPNZD
- NZDCAD
- NZDCHF
- NZDHKD
- NZDJPY
- NZDSGD
- NZDUSD

Market Hours

EURNZD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/NZD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/NZD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the EUR/NZD pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the EUR/NZD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the EUR/NZD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The EUR/NZD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

GBPNZD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/NZD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/NZD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the GBP/NZD pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the GBP/NZD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the GBP/NZD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The GBP/NZD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDCAD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/CAD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/CAD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NZD/CAD pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the NZD/CAD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the NZD/CAD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The NZD/CAD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDCHF

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/CHF pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/CHF pair in the Forex market:

Weekday	Time (America/New York)
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NZD/CHF pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the NZD/CHF pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the NZD/CHF pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The NZD/CHF pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDHKD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/HKD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/HKD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NZD/HKD pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the NZD/HKD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the NZD/HKD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The NZD/HKD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/JPY pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/JPY pair in the Forex market:

Weekday	Time (America/New York)
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NZD/JPY pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the NZD/JPY pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the NZD/JPY pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The NZD/JPY pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDSGD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/SGD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/SGD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NZD/SGD pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the NZD/SGD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the NZD/SGD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The NZD/SGD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDUSD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/USD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/USD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NZD/USD pair in the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31	2024-01-01	

Early Closes

The following table shows the early closes for the NZD/USD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00

Late Opens

The following table shows the late opens for the NZD/USD pair in the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00

Time Zone

The NZD/USD pair in the Forex market trades in the `America/New York` time zone.

Asset Classes

Futures

Select one of the following Futures markets to see its operating hours:

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateFuturesAlgorithm.py
BasicTemplateFuturesAlgorithm.cs

Futures

Requesting Data

Introduction

Request Futures data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Futures dataset listing . To trade Futures live, you can use our Futures data feed or one of the brokerage data feeds .

Create Subscriptions

Before you can subscribe to a Futures contract, you must get the contract `Symbol` .

Get Contract Symbols

To get Futures contract `Symbol` objects, call the `CreateFuture` method or use the `FutureChainProvider` . If you use the `CreateFuture` method, you need to know the specific contract details.

```
PY
self.contract_symbol = Symbol.CreateFuture(Futures.Indices.SP500EMini,
    Market.CME, datetime(2022,6,17))
```

If you use the `FutureChainProvider` , you need to create the continuous contract `Symbol` of the Future. The `GetFutureContractList` method of `FutureChainProvider` returns a list of `Symbol` objects that reference the available Futures contracts for a given underlying Future on a given date.

```
PY
continuous_future_symbol = Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME)
contract_symbols = self.FutureChainProvider.GetFutureContractList(continuous_future_symbol, self.Time)
self.contract_symbol = sorted(contract_symbols, key=lambda symbol: symbol.ID.Date)[0]
```

Subscribe to Contracts

To create a Futures contract subscription, pass the contract `Symbol` to the `AddFutureContract` method. Save a reference to the contract `Symbol` so you can easily access the contract in the FuturesChain that LEAN passes to the `OnData` method.

```
PY
self.AddFutureContract(self.contract_symbol)
```

The `AddFutureContract` method creates a subscription for a single Future contract and adds it to your **user-defined** universe. To create a dynamic universe of Futures contracts, add a Futures universe or a Futures Universe Selection model .

Warm Up Contract Prices

If you subscribe to a Futures contract with `AddFutureContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel,
    FuncSecuritySeeder(self.GetLastKnownPrices)));
```

PY

Supported Assets

To view the supported assets in the US Futures dataset, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Futures subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Futures contract subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddFutureContract` method.

```
self.AddFutureContract(self.contract_symbol, Resolution.Daily)
```

PY

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Futures:

You don't need to pass a `market` argument to the `AddFutureContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.AddFutureContract(self.contract_symbol, fillForward=False)
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Futures are already leveraged products, so you can't change their leverage with the default margin model .

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.AddFutureContract(self.contract_symbol, extendedMarketHours=True)
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

In general, we model most Futures market hours with the following segments:

Market Segment	Time
Pre-market	00:00:00 to 09:30:00
Market	09:30:00 to 17:00:00
Post-market	18:00:00 to 00:00:00

We model it this way because some Futures, like VIX, have pre- and post-market hours, so we standardized it. With this segmentation, if you set a Scheduled Events for the market open, it's set for 9:30 AM instead of midnight.

Continuous Contracts

A continuous Futures contract represents a single contract that maps to other contracts over time as the rollover rules are met. For more information about continuous Futures contracts, see Continuous Contracts .

Properties

The `AddFutureContract` method returns a `Future` object, which have the following properties:

Futures

Handling Data

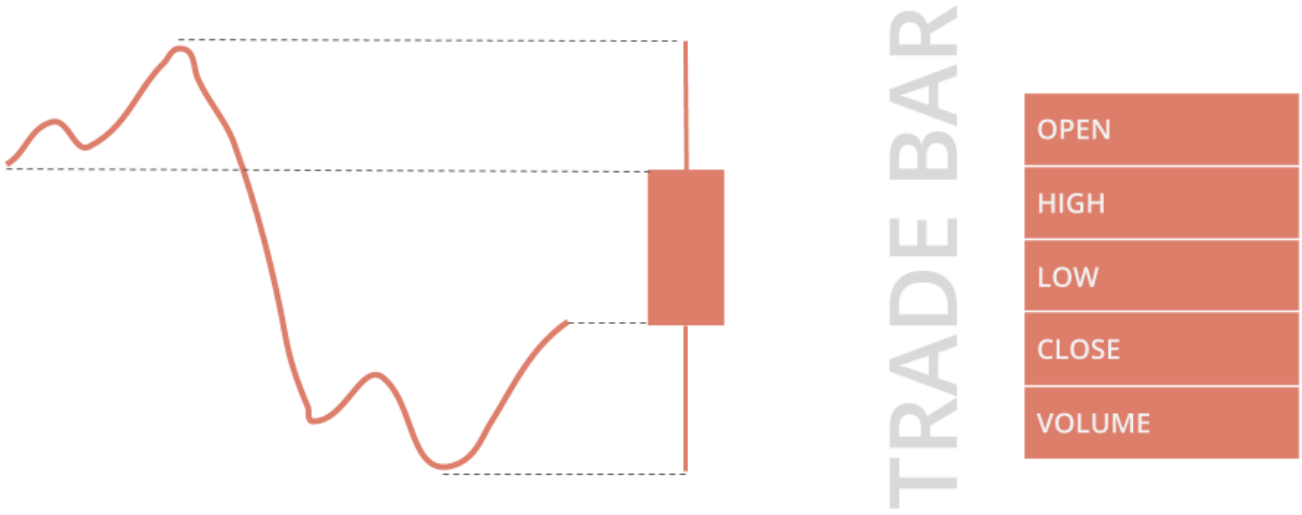
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the contract ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

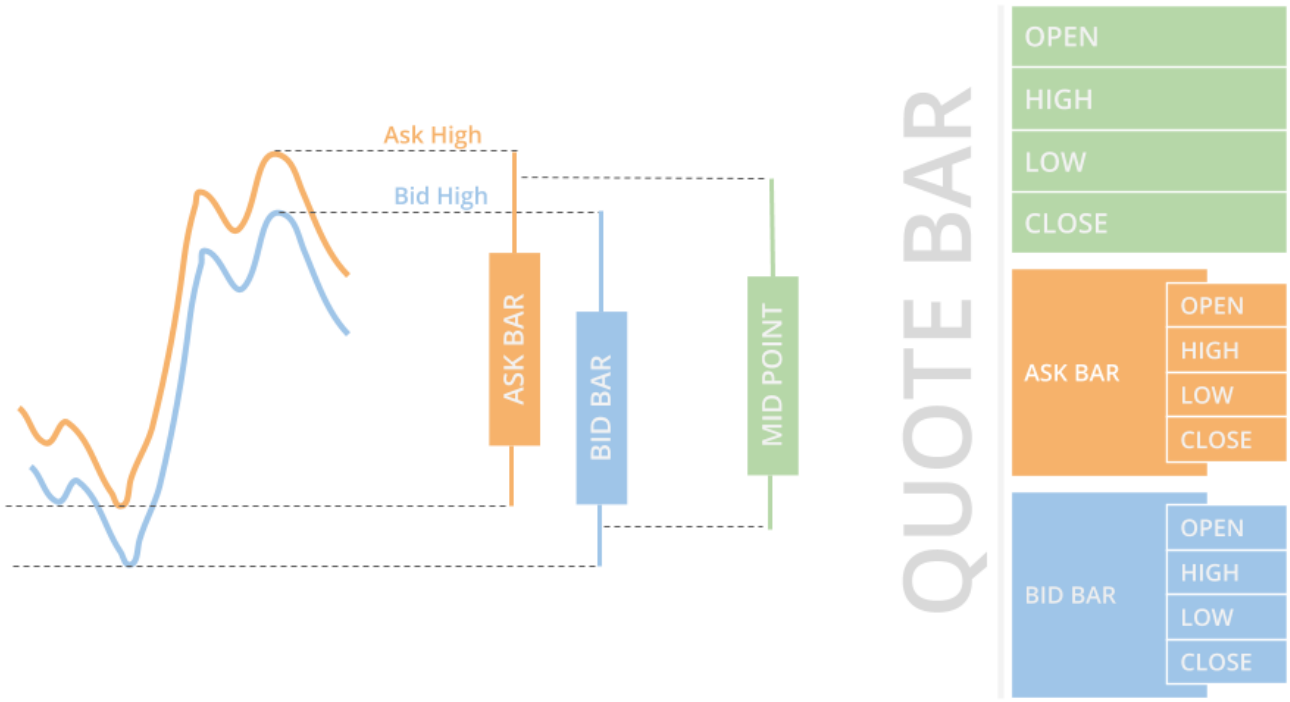
```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.Bars:
        trade_bar = slice.Bars[self.contract_symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively get quotes or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.contract_symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the contract. A quote tick is an offer to buy or sell the contract at a specific price. `Tick` objects have the following properties:

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.Ticks:
        ticks = slice.Ticks[self.contract_symbol]
        for tick in ticks:
            price = tick.Price
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `list[Tick]` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            price = tick.Price
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able

to perform your own online tick filtering.

Futures Chains

`FuturesChain` objects represent an entire chain of contracts for a single underlying Future. They have the following properties:

To get the `FuturesChain` , index the `FuturesChains` property of the `Slice` with the continuous contract `Symbol` .

```
PY
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.contract_symbol.Canonical)
    if chain:
        contracts = chain.Contracts
```

You can also loop through the `FuturesChains` property to get each `FuturesChain` .

```
PY
def OnData(self, slice: Slice) -> None:
    for continuous_contract_symbol, chain in slice.FuturesChains.items():
        contracts = chain.Contracts
```

Futures Contracts

`FuturesContract` objects represent the data of a single Futures contract in the market. They have the following properties:

To get the Futures contracts in the `Slice` , use the `Contracts` property of the `FuturesChain` .

```
PY
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            price = contract.LastPrice
```

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Future` or `FutureContract` .

```
PY
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            open_interest = contract.OpenInterest
```

Symbol Changes

When the continuous contract rolls over, LEAN passes a `SymbolChangedEvent` to your `OnData` method, which contains the old contract `Symbol` and the new contract `Symbol` . `SymbolChangedEvent` objects have the following properties:

To get the `SymbolChangedEvent` , use the `SymbolChangedEvents` property of the `Slice` .

```py
def OnData(self, slice: Slice) -> None:
    for changed_event in slice.SymbolChangedEvents.Values:
        self.Log(f"Contract rollover from {changed_event.OldSymbol} to {changed_event.NewSymbol}")
```

Futures

Market Hours

Market Hours

CBOT

Introduction

This page shows the trading hours, holidays, and time zone of the CBOT Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25

Date (*yyyy-mm-dd*)				
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28

Date (*yyyy-mm-dd*)				
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CBOT Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CBOT Future market:

Symbol	Name
10Y	Micro 10-Year Yield Futures
2YY	Micro 2-Year Yield Futures
30Y	Micro 30-Year Yield Futures
5YY	Micro 5-Year Yield Futures
AW	Bloomberg Commodity Index Futures
BCF	Black Sea Corn Financially Settled (Platts) Futures
BWF	Black Sea Wheat Financially Settled (Platts) Futures
EH	Ethanol Futures
F1U	5-Year USD MAC Swap Futures
KE	KC HRW Wheat Futures
MYM	Micro E-mini Dow Jones Industrial Average Index Futures
TN	Ultra 10-Year U.S. Treasury Note Futures
UB	Ultra U.S. Treasury Bond Futures
YM	E-mini Dow ($5) Futures
ZB	U.S. Treasury Bond Futures
ZC	Corn Futures
ZF	5-Year T-Note Futures
ZL	Soybean Oil Futures
ZM	Soybean Meal Futures
ZN	10-Year T-Note Futures
ZO	Oats Futures
ZS	Soybean Futures
ZT	2-Year T-Note Futures
ZW	Chicago SRW Wheat Futures

CBOT

10Y

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 10-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2008-12-31	15:15:00
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

The following table shows the late opens for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2009-01-02	05:00:00
2011-12-27	05:00:00
2012-01-03	05:00:00
2012-12-26	05:00:00
2013-01-02	05:00:00
2013-12-26	05:00:00
2014-01-02	05:00:00

Time Zone

The Micro 10-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

2YY

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 2-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2008-12-31	15:15:00
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

The following table shows the late opens for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2009-01-02	05:00:00
2011-12-27	05:00:00
2012-01-03	05:00:00
2012-12-26	05:00:00
2013-01-02	05:00:00
2013-12-26	05:00:00
2014-01-02	05:00:00

Time Zone

The Micro 2-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

30Y

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 30-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2008-12-31	15:15:00
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

The following table shows the late opens for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2009-01-02	05:00:00
2011-12-27	05:00:00
2012-01-03	05:00:00
2012-12-26	05:00:00
2013-01-02	05:00:00
2013-12-26	05:00:00
2014-01-02	05:00:00

Time Zone

The Micro 30-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

5YY

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 5-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2008-12-31	15:15:00
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

The following table shows the late opens for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2009-01-02	05:00:00
2011-12-27	05:00:00
2012-01-03	05:00:00
2012-12-26	05:00:00
2013-01-02	05:00:00
2013-12-26	05:00:00
2014-01-02	05:00:00

Time Zone

The Micro 5-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

AW

Introduction

This page shows the trading hours, holidays, and time zone of the Bloomberg Commodity Index Futures contract in the CBOT Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Bloomberg Commodity Index Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:15:00 to 13:30:00
Tuesday	08:15:00 to 13:30:00
Wednesday	08:15:00 to 13:30:00
Thursday	08:15:00 to 13:30:00
Friday	08:15:00 to 13:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Bloomberg Commodity Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2011-11-25	12:00:00
2011-12-31	12:00:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-11-27	12:00:00
2015-12-24	12:00:00
2016-11-25	12:00:00
2016-12-23	12:00:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-22	12:00:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2019-07-03	12:00:00
2019-11-29	12:00:00
2019-12-24	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The Bloomberg Commodity Index Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

BCF

Introduction

This page shows the trading hours, holidays, and time zone of the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:45:00
2018-12-24	12:45:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:45:00
2019-12-24	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

BWF

Introduction

This page shows the trading hours, holidays, and time zone of the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:45:00
2018-12-24	12:45:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:45:00
2019-12-24	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

EH

Introduction

This page shows the trading hours, holidays, and time zone of the Ethanol Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ethanol Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ethanol Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ethanol Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Ethanol Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2011-11-25	12:00:00
2011-12-31	12:00:00
2013-05-27	12:15:00
2013-07-04	12:15:00
2013-09-02	12:15:00
2013-11-28	12:15:00
2013-11-29	12:45:00
2013-12-24	12:45:00
2014-01-20	12:15:00
2014-02-17	12:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-12-24	12:45:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:45:00
2015-12-24	12:45:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:45:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:45:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-09-03	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-11-22	12:00:00
2018-11-23	12:45:00
2018-12-24	12:45:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:45:00
2019-12-24	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Ethanol Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

F1U

Introduction

This page shows the trading hours, holidays, and time zone of the 5-Year USD MAC Swap Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 5-Year USD MAC Swap Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 5-Year USD MAC Swap Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 5-Year USD MAC Swap Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The 5-Year USD MAC Swap Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

KE

Introduction

This page shows the trading hours, holidays, and time zone of the KC HRW Wheat Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the KC HRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the KC HRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the KC HRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the KC HRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2013-01-01	2013-01-20	2013-02-17	2013-03-29	2013-05-26
2013-07-04	2013-09-01	2013-11-28	2013-12-25	2014-01-01
2014-01-19	2014-02-16	2014-04-18	2014-05-25	2014-07-04
2014-07-06	2014-08-31	2014-11-27	2014-12-25	2015-01-01
2015-01-18	2015-02-15	2015-04-03	2015-05-24	2015-07-03
2015-09-06	2015-11-26	2015-12-25	2016-01-01	2016-01-17
2016-02-14	2016-03-25	2016-05-29	2016-07-03	2016-07-04
2016-09-04	2016-11-24	2017-01-15	2017-02-19	2017-04-14
2017-05-28	2017-07-04	2017-09-03	2017-11-23	2017-12-25
2018-01-01	2018-01-14	2018-02-18	2018-03-30	2018-05-27
2018-07-04	2018-09-02	2018-11-22	2018-12-25	2019-01-01
2019-01-20	2019-02-17	2019-04-19	2019-05-26	2019-07-04
2019-09-01	2019-11-28	2019-12-25	2020-01-01	2020-01-19
2020-02-16	2020-04-10	2020-05-24	2020-07-03	2020-09-06
2020-11-26	2020-12-25	2021-01-01	2022-12-25	2022-12-26
2023-01-01	2023-01-02	2023-01-16	2023-02-20	2023-04-07
2023-05-29	2023-06-19	2023-07-04	2023-09-04	2023-11-23
2023-12-25	2024-01-01			

Early Closes

The following table shows the early closes for the KC HRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2013-05-24	13:15:00
2013-07-03	12:00:00
2013-08-30	13:15:00
2013-11-27	13:15:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2013-12-26	13:15:00
2013-12-31	13:15:00
2014-01-02	13:15:00
2014-01-17	13:15:00
2014-02-14	13:15:00
2014-04-17	13:15:00
2014-05-23	13:15:00
2014-07-03	12:00:00
2014-08-29	13:15:00
2014-11-26	13:15:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2014-12-26	13:15:00
2014-12-31	13:15:00
2015-01-02	13:15:00
2015-01-16	13:15:00
2015-02-13	13:15:00
2015-04-02	13:15:00
2015-05-22	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2015-07-02	12:00:00
2015-09-04	13:20:00
2015-11-25	13:20:00
2015-11-27	12:05:00
2015-12-28	13:20:00
2015-12-31	13:20:00
2016-01-04	13:20:00
2016-01-15	13:20:00
2016-02-12	13:20:00
2016-03-24	13:20:00
2016-05-27	13:20:00
2016-07-01	13:20:00
2016-09-02	13:20:00
2016-11-23	13:20:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2016-12-27	13:20:00
2016-12-30	13:20:00
2017-01-03	13:20:00
2017-01-13	13:20:00
2017-02-17	13:20:00
2017-04-13	13:20:00
2017-05-26	13:20:00
2017-07-03	12:05:00
2017-09-01	13:20:00
2017-11-22	13:20:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2017-11-24	12:05:00
2017-12-22	12:05:00
2017-12-26	13:20:00
2017-12-29	13:20:00
2018-01-02	13:20:00
2018-01-12	13:20:00
2018-02-16	13:20:00
2018-03-29	13:20:00
2018-05-25	13:20:00
2018-07-03	12:05:00
2018-08-31	13:20:00
2018-11-21	13:20:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2018-12-26	13:20:00
2018-12-31	13:20:00
2019-01-02	13:20:00
2019-01-18	13:20:00
2019-02-15	13:20:00
2019-04-18	13:20:00
2019-05-24	13:20:00
2019-07-03	12:05:00
2019-08-30	13:20:00
2019-11-27	13:20:00
2019-11-29	12:05:00
2019-12-24	12:05:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-26	13:20:00
2019-12-31	13:20:00
2020-01-02	13:20:00
2020-01-17	13:20:00
2020-02-14	13:20:00
2020-04-09	13:20:00
2020-05-22	13:20:00
2020-07-02	12:05:00
2020-09-04	13:20:00
2020-11-25	13:20:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2020-12-28	13:20:00
2020-12-31	13:20:00
2021-01-04	13:20:00

Late Opens

The following table shows the late opens for the KC HRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2013-01-21	19:00:00
2013-02-22	19:00:00
2013-05-27	19:00:00
2013-07-05	08:30:00
2013-09-02	19:00:00
2013-11-29	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2014-02-17	19:00:00
2014-04-20	19:00:00
2014-05-26	19:00:00
2014-07-07	08:30:00
2014-09-01	19:00:00
2014-11-28	08:30:00
2014-12-26	08:30:00
2015-01-02	08:30:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-04-05	19:00:00
2015-05-25	19:00:00
2015-07-05	19:00:00
2015-07-06	08:30:00
2015-09-07	19:00:00
2015-11-27	08:30:00
2015-12-27	19:00:00
2015-12-28	08:30:00
2016-01-03	19:00:00
2016-01-04	08:30:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-03-27	19:00:00
2016-05-30	19:00:00
2016-07-05	08:30:00
2016-09-05	19:00:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2016-11-25	08:30:00
2016-12-26	19:00:00
2016-12-27	08:30:00
2017-01-02	19:00:00
2017-01-03	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-04-16	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2017-12-26	08:30:00
2018-01-02	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-02-20	08:30:00
2018-04-01	19:00:00
2018-05-28	19:00:00
2018-05-29	08:30:00
2018-07-05	08:30:00
2018-09-03	19:00:00
2018-11-23	08:30:00
2018-12-26	08:30:00
2019-01-02	08:30:00
2019-01-21	19:00:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2019-01-22	08:30:00
2019-02-18	19:00:00
2019-02-19	08:30:00
2019-04-21	19:00:00
2019-05-27	19:00:00
2019-05-28	08:30:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-09-03	08:30:00
2019-11-29	08:30:00
2019-12-26	08:30:00
2020-01-02	08:30:00
2020-01-20	19:00:00
2020-01-21	08:30:00
2020-02-17	19:00:00
2020-02-18	08:30:00
2020-04-12	19:00:00
2020-05-25	19:00:00
2020-05-26	08:30:00
2020-07-05	19:00:00
2020-07-06	08:30:00
2020-09-07	19:00:00
2020-09-08	08:30:00
2020-11-27	08:30:00
2020-12-27	19:00:00
2020-12-28	08:30:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2021-01-03	19:00:00
2021-01-04	08:30:00

Time Zone

The KC HRW Wheat Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

MYM

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market trades in the `America/New York` time zone.

CBOT

TN

Introduction

This page shows the trading hours, holidays, and time zone of the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

UB

Introduction

This page shows the trading hours, holidays, and time zone of the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-24	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Ultra U.S. Treasury Bond Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

YM

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Dow ($5) Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini Dow ($5) Futures contract in the CBOT Future market trades in the `America/New York` time zone.

CBOT

ZB

Introduction

This page shows the trading hours, holidays, and time zone of the U.S. Treasury Bond Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The U.S. Treasury Bond Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZC

Introduction

This page shows the trading hours, holidays, and time zone of the Corn Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Corn Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Corn Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Corn Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Corn Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZF

Introduction

This page shows the trading hours, holidays, and time zone of the 5-Year T-Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 5-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 5-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 5-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The 5-Year T-Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZL

Introduction

This page shows the trading hours, holidays, and time zone of the Soybean Oil Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Soybean Oil Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Soybean Oil Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Soybean Oil Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Soybean Oil Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZM

Introduction

This page shows the trading hours, holidays, and time zone of the Soybean Meal Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Soybean Meal Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Soybean Meal Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Soybean Meal Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Soybean Meal Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZN

Introduction

This page shows the trading hours, holidays, and time zone of the 10-Year T-Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 10-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 10-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 10-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The 10-Year T-Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZO

Introduction

This page shows the trading hours, holidays, and time zone of the Oats Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Oats Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Oats Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Oats Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Oats Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZS

Introduction

This page shows the trading hours, holidays, and time zone of the Soybean Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Soybean Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Soybean Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Soybean Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Soybean Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZT

Introduction

This page shows the trading hours, holidays, and time zone of the 2-Year T-Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 2-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 2-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 2-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The 2-Year T-Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZW

Introduction

This page shows the trading hours, holidays, and time zone of the Chicago SRW Wheat Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Chicago SRW Wheat Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

Market Hours

CFE

Introduction

This page shows the trading hours, holidays, and time zone of the CFE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CFE Future market:

Weekday	Time (America/Chicago)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 24:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CFE Future market trades in the `America/Chicago` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CFE Future market:

Symbol	Name
VX	VIX futures

CFE

VX

Introduction

This page shows the trading hours, holidays, and time zone of the VIX futures contract in the CFE Future market.

Pre-market Hours

The following table shows the pre-market hours for the VIX futures contract in the CFE Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the VIX futures contract in the CFE Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

The following table shows the post-market hours for the VIX futures contract in the CFE Future market:

Weekday	Time (America/Chicago)
Monday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	15:00:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the CFE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31

Date (*yyyy-mm-dd*)				
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX futures contract in the CFE Future market trades in the `America/Chicago` time zone.

Market Hours

CME

Introduction

This page shows the trading hours, holidays, and time zone of the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CME Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CME Future market:

Symbol	Name
6A	Australian Dollar Futures
6B	British Pound Futures
6C	Canadian Dollar Futures
6E	Euro FX Futures
6J	Japanese Yen Futures
6L	Brazilian Real Futures
6M	Mexican Peso Futures
6N	New Zealand Dollar Futures
6R	Russian Ruble Futures
6S	Swiss Franc Futures
6Z	South African Rand Futures
ACD	Australian Dollar/Canadian Dollar Futures
AJY	Australian Dollar/Japanese Yen Futures
ANE	Australian Dollar/New Zealand Dollar Futures
BIO	E-mini Nasdaq-100 Biotechnology Index Futures
BTC	Bitcoin Futures
CB	Cash-settled Butter Futures
CJY	Canadian Dollar/Japanese Yen Futures
CNH	Standard-Size USD/Offshore RMB (CNH) Futures
CSC	Cash-Settled Cheese Futures
DC	Class III Milk Futures
DY	Dry Whey Futures
E7	E-mini Euro FX Futures
EAD	Euro/Australian Dollar Futures

Symbol	Name
ECD	Euro/Canadian Dollar Futures
EI	E-mini FTSE Emerging Index Futures
EMD	E-mini S&P MidCap 400 Futures
ES	E-mini S&P 500 Futures
ESK	Euro/Swedish Krona Futures
ETH	Ether Futures
GD	S&P-GSCI Commodity Index Futures
GDK	Class IV Milk Futures
GE	Eurodollar Futures
GF	Feeder Cattle Futures
GNF	Nonfat Dry Milk Futures
HE	Lean Hog Futures
IBV	USD-Denominated Ibovespa Index Futures
J7	E-mini Japanese Yen Futures
LBS	Random Length Lumber Futures
LE	Live Cattle Futures
M2K	Micro E-mini Russell 2000 Index Futures
M6A	Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures
M6B	Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures
M6C	Micro USD/CAD Futures
M6E	Micro Euro/U.S. Dollar (EUR/USD) Futures
M6J	Micro USD/JPY Futures
M6S	Micro USD/CHF Futures
MBT	Micro Bitcoin Futures
MCD	Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures

Symbol	Name
MES	Micro E-mini Standard and Poor's 500 Stock Price Index Futures
MET	Micro Ether Futures
MIR	Micro INR/USD Futures
MJY	Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures
MNH	Micro USD/CNH Futures
MNQ	Micro E-mini Nasdaq-100 Index Futures
MSF	Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures
NKD	Nikkei/USD Futures
NQ	E-mini Nasdaq-100 Futures
RTY	E-mini Russell 2000 Index Futures

CME

6A

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Australian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6B

Introduction

This page shows the trading hours, holidays, and time zone of the British Pound Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the British Pound Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the British Pound Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the British Pound Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the British Pound Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The British Pound Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6C

Introduction

This page shows the trading hours, holidays, and time zone of the Canadian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Canadian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6E

Introduction

This page shows the trading hours, holidays, and time zone of the Euro FX Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Euro FX Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6J

Introduction

This page shows the trading hours, holidays, and time zone of the Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6L

Introduction

This page shows the trading hours, holidays, and time zone of the Brazilian Real Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Brazilian Real Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Brazilian Real Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Brazilian Real Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Brazilian Real Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Brazilian Real Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6M

Introduction

This page shows the trading hours, holidays, and time zone of the Mexican Peso Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mexican Peso Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mexican Peso Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Mexican Peso Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mexican Peso Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Mexican Peso Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6N

Introduction

This page shows the trading hours, holidays, and time zone of the New Zealand Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The New Zealand Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6R

Introduction

This page shows the trading hours, holidays, and time zone of the Russian Ruble Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Russian Ruble Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Russian Ruble Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Russian Ruble Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Russian Ruble Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Russian Ruble Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6S

Introduction

This page shows the trading hours, holidays, and time zone of the Swiss Franc Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Swiss Franc Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Swiss Franc Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Swiss Franc Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Swiss Franc Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Swiss Franc Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6Z

Introduction

This page shows the trading hours, holidays, and time zone of the South African Rand Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the South African Rand Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the South African Rand Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the South African Rand Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the South African Rand Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The South African Rand Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ACD

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar/Canadian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Australian Dollar/Canadian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

AJY

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar/Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Australian Dollar/Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ANE

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Australian Dollar/New Zealand Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

BIO

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

BTC

Introduction

This page shows the trading hours, holidays, and time zone of the Bitcoin Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Bitcoin Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CB

Introduction

This page shows the trading hours, holidays, and time zone of the Cash-settled Butter Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Cash-settled Butter Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Cash-settled Butter Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Cash-settled Butter Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Cash-settled Butter Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2011-04-21	13:55:00
2011-12-31	12:15:00
2012-12-24	12:00:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-12-24	12:00:00
2014-04-17	13:55:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2014-12-26	13:55:00
2015-01-02	13:55:00
2015-03-24	13:55:00
2015-04-02	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-12-23	12:00:00
2016-12-30	13:55:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-12-22	12:00:00
2017-12-31	13:55:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-04-09	13:55:00
2020-07-02	12:00:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-04-01	13:55:00
2022-04-14	13:55:00

Late Opens

There are no days with late opens.

Time Zone

The Cash-settled Butter Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CJY

Introduction

This page shows the trading hours, holidays, and time zone of the Canadian Dollar/Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Canadian Dollar/Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CNH

Introduction

This page shows the trading hours, holidays, and time zone of the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CSC

Introduction

This page shows the trading hours, holidays, and time zone of the Cash-Settled Cheese Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Cash-Settled Cheese Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Cash-Settled Cheese Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Cash-Settled Cheese Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Cash-Settled Cheese Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2011-04-21	13:55:00
2011-12-31	12:15:00
2012-12-24	12:00:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-12-24	12:00:00
2014-04-17	13:55:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2014-12-26	13:55:00
2015-01-02	13:55:00
2015-03-24	13:55:00
2015-04-02	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-12-23	12:00:00
2016-12-30	13:55:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-12-22	12:00:00
2017-12-31	13:55:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-04-09	13:55:00
2020-07-02	12:00:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-04-01	13:55:00
2022-04-14	13:55:00

Late Opens

There are no days with late opens.

Time Zone

The Cash-Settled Cheese Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

DC

Introduction

This page shows the trading hours, holidays, and time zone of the Class III Milk Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Class III Milk Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Class III Milk Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Class III Milk Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Class III Milk Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2011-04-21	13:55:00
2011-12-31	12:15:00
2012-12-24	12:00:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-12-24	12:00:00
2014-04-17	13:55:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2014-12-26	13:55:00
2015-01-02	13:55:00
2015-03-24	13:55:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-12-23	12:00:00
2016-12-30	13:55:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-12-22	12:00:00
2017-12-31	13:55:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-04-09	13:55:00
2020-07-02	12:00:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-04-01	13:55:00
2022-04-14	13:55:00

Late Opens

There are no days with late opens.

Time Zone

The Class III Milk Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

DY

Introduction

This page shows the trading hours, holidays, and time zone of the Dry Whey Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Dry Whey Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Dry Whey Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 13:55:00

Post-market Hours

The following table shows the post-market hours for the Dry Whey Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Dry Whey Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2011-04-21	13:55:00
2011-12-31	12:15:00
2012-12-24	12:00:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-12-24	12:00:00
2014-04-17	13:55:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2014-12-26	13:55:00
2015-01-02	13:55:00
2015-03-24	13:55:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-12-23	12:00:00
2016-12-30	13:55:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-12-22	12:00:00
2017-12-31	13:55:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-04-09	13:55:00
2020-07-02	12:00:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-04-01	13:55:00
2022-04-14	13:55:00

Late Opens

There are no days with late opens.

Time Zone

The Dry Whey Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

E7

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Euro FX Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini Euro FX Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

EAD

Introduction

This page shows the trading hours, holidays, and time zone of the Euro/Australian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro/Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro/Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro/Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Euro/Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Euro/Australian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ECD

Introduction

This page shows the trading hours, holidays, and time zone of the Euro/Canadian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Euro/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Euro/Canadian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

EI

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini FTSE Emerging Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini FTSE Emerging Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

EMD

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini S&P MidCap 400 Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini S&P MidCap 400 Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ES

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini S&P 500 Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini S&P 500 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini S&P 500 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini S&P 500 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini S&P 500 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini S&P 500 Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

ESK

Introduction

This page shows the trading hours, holidays, and time zone of the Euro/Swedish Krona Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro/Swedish Krona Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro/Swedish Krona Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro/Swedish Krona Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Euro/Swedish Krona Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Euro/Swedish Krona Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ETH

Introduction

This page shows the trading hours, holidays, and time zone of the Ether Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Ether Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GD

Introduction

This page shows the trading hours, holidays, and time zone of the S&P-GSCI Commodity Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The S&P-GSCI Commodity Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GDK

Introduction

This page shows the trading hours, holidays, and time zone of the Class IV Milk Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Class IV Milk Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Class IV Milk Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Class IV Milk Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Class IV Milk Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2011-04-21	13:55:00
2011-12-31	12:15:00
2012-12-24	12:00:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-12-24	12:00:00
2014-04-17	13:55:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2014-12-26	13:55:00
2015-01-02	13:55:00
2015-03-24	13:55:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-12-23	12:00:00
2016-12-30	13:55:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-12-22	12:00:00
2017-12-31	13:55:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-04-09	13:55:00
2020-07-02	12:00:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-04-01	13:55:00
2022-04-14	13:55:00

Late Opens

There are no days with late opens.

Time Zone

The Class IV Milk Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GE

Introduction

This page shows the trading hours, holidays, and time zone of the Eurodollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Eurodollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Eurodollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Eurodollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Eurodollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2008-12-31	15:15:00
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

The following table shows the late opens for the Eurodollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-02	05:00:00
2011-12-27	05:00:00
2012-01-03	05:00:00
2012-12-26	05:00:00
2013-01-02	05:00:00
2013-12-26	05:00:00
2014-01-02	05:00:00

Time Zone

The Eurodollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GF

Introduction

This page shows the trading hours, holidays, and time zone of the Feeder Cattle Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Feeder Cattle Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:05:00
Tuesday	08:30:00 to 13:05:00
Wednesday	08:30:00 to 13:05:00
Thursday	08:30:00 to 13:05:00
Friday	08:30:00 to 13:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Feeder Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2010-12-31	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-04-21	13:55:00
2011-11-25	12:15:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-07-03	12:15:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-04-17	13:55:00
2014-07-03	12:15:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-04-02	13:55:00
2015-07-02	12:15:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2015-12-31	13:55:00
2016-11-25	12:15:00
2016-12-23	12:15:00
2017-07-03	12:15:00
2017-11-24	12:15:00
2017-12-22	12:15:00
2018-07-03	12:15:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-07-03	12:15:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-07-02	12:15:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00
2022-11-25	12:05:00

Late Opens

The following table shows the late opens for the Feeder Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2011-11-24	17:00:00
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-17	09:05:00
2012-02-21	09:05:00
2012-05-29	09:05:00
2012-07-05	09:05:00
2012-09-04	09:05:00
2012-11-22	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-22	09:05:00
2013-02-19	09:05:00
2013-05-28	09:05:00
2013-07-05	09:05:00
2013-09-03	09:05:00
2013-11-29	09:05:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-21	09:05:00
2014-02-18	09:05:00
2014-04-21	09:05:00
2014-05-27	09:05:00
2014-07-07	09:05:00
2014-09-02	09:05:00
2015-01-20	09:05:00
2015-02-17	09:05:00
2015-04-06	09:05:00
2015-05-26	09:05:00
2015-07-06	09:05:00
2015-09-08	09:05:00
2015-12-28	09:05:00
2016-01-04	09:05:00
2016-01-19	09:05:00
2016-02-16	09:05:00

Time Zone

The Feeder Cattle Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GNF

Introduction

This page shows the trading hours, holidays, and time zone of the Nonfat Dry Milk Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nonfat Dry Milk Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 13:55:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21

Date (*yyyy-mm-dd*)				
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25

Date (*yyyy-mm-dd*)				
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Nonfat Dry Milk Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2011-04-21	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-12-31	12:15:00
2012-12-24	12:00:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-12-24	12:00:00
2014-04-17	13:55:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2014-12-26	13:55:00
2015-01-02	13:55:00
2015-03-24	13:55:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-12-23	12:00:00
2016-12-30	13:55:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-12-22	12:00:00
2017-12-31	13:55:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-07-03	12:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-04-09	13:55:00
2020-07-02	12:00:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-04-01	13:55:00
2022-04-14	13:55:00

Late Opens

There are no days with late opens.

Time Zone

The Nonfat Dry Milk Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

HE

Introduction

This page shows the trading hours, holidays, and time zone of the Lean Hog Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Lean Hog Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:05:00
Tuesday	08:30:00 to 13:05:00
Wednesday	08:30:00 to 13:05:00
Thursday	08:30:00 to 13:05:00
Friday	08:30:00 to 13:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Lean Hog Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-07-03	12:15:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-04-17	13:55:00
2014-07-03	12:15:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-04-02	13:55:00
2015-07-02	12:15:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2015-12-31	13:55:00
2016-11-25	12:15:00
2016-12-23	12:15:00
2017-07-03	12:15:00
2017-11-24	12:15:00
2017-12-22	12:15:00
2018-07-03	12:15:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-07-03	12:15:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-07-02	12:15:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00
2022-11-25	12:05:00

Late Opens

The following table shows the late opens for the Lean Hog Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2011-11-24	17:00:00
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-17	09:05:00
2012-02-21	09:05:00
2012-05-29	09:05:00
2012-07-05	09:05:00
2012-09-04	09:05:00
2012-11-22	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-22	09:05:00
2013-02-19	09:05:00
2013-05-28	09:05:00
2013-07-05	09:05:00
2013-09-03	09:05:00
2013-11-29	09:05:00
2013-12-26	09:05:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2014-01-02	09:05:00
2014-01-21	09:05:00
2014-02-18	09:05:00
2014-04-21	09:05:00
2014-05-27	09:05:00
2014-07-07	09:05:00
2014-09-02	09:05:00
2015-01-20	09:05:00
2015-02-17	09:05:00
2015-04-06	09:05:00
2015-05-26	09:05:00
2015-07-06	09:05:00
2015-09-08	09:05:00
2015-12-28	09:05:00
2016-01-04	09:05:00
2016-01-19	09:05:00
2016-02-16	09:05:00

Time Zone

The Lean Hog Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

IBV

Introduction

This page shows the trading hours, holidays, and time zone of the USD-Denominated Ibovespa Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The USD-Denominated Ibovespa Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

J7

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

LBS

Introduction

This page shows the trading hours, holidays, and time zone of the Random Length Lumber Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Random Length Lumber Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	06:00:00 to 09:00:00
Tuesday	06:00:00 to 09:00:00
Wednesday	06:00:00 to 09:00:00
Thursday	06:00:00 to 09:00:00
Friday	06:00:00 to 09:00:00

Regular Trading Hours

The following table shows the regular trading hours for the Random Length Lumber Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	09:00:00 to 15:05:00
Tuesday	09:00:00 to 15:05:00
Wednesday	09:00:00 to 15:05:00
Thursday	09:00:00 to 15:05:00
Friday	09:00:00 to 15:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Random Length Lumber Futures contract in the CME Future

market:

Date (*yyyy-mm-dd*)				
2013-01-01	2013-01-21	2013-02-18	2013-03-29	2013-05-27
2013-07-04	2013-09-02	2013-11-28	2013-12-25	2014-01-01
2014-01-20	2014-02-17	2014-04-18	2014-04-21	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-01	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-25
2019-01-01	2019-01-21	2019-02-18	2019-04-19	2019-05-27
2019-07-04	2019-09-02	2019-11-28	2019-12-25	2020-01-01
2020-01-20	2020-02-17	2020-04-10	2020-05-25	2020-07-03
2020-09-07	2020-11-26	2020-12-25	2021-01-01	2021-01-17
2021-01-18	2021-02-14	2021-02-15	2021-04-02	2021-05-30
2021-05-31	2021-07-04	2021-07-05	2021-09-05	2021-09-06
2021-11-24	2021-11-25	2021-12-24	2021-12-26	2022-01-16
2022-01-17	2022-02-20	2022-02-21	2022-04-15	2022-05-29
2022-05-30	2022-06-19	2022-06-20	2022-07-03	2022-07-04
2022-09-04	2022-09-05	2022-11-23	2022-11-24	2022-12-25
2022-12-26	2023-01-01	2023-01-02	2023-01-16	2023-02-20
2023-04-07	2023-05-29	2023-06-19	2023-07-04	2023-09-04
2023-11-23	2023-12-25	2024-01-01		

Early Closes

The following table shows the early closes for the Random Length Lumber Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2013-07-03	12:00:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-07-03	12:00:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-07-02	12:00:00
2015-11-27	12:00:00
2015-12-24	12:00:00
2016-11-27	12:00:00
2016-12-24	12:00:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-24	12:00:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2019-07-03	12:00:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-07-02	12:05:00
2020-11-26	12:05:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2022-11-25	12:05:00

Late Opens

There are no days with late opens.

Time Zone

The Random Length Lumber Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

LE

Introduction

This page shows the trading hours, holidays, and time zone of the Live Cattle Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Live Cattle Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:05:00
Tuesday	08:30:00 to 13:05:00
Wednesday	08:30:00 to 13:05:00
Thursday	08:30:00 to 13:05:00
Friday	08:30:00 to 13:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Live Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-07-03	12:15:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-04-17	13:55:00
2014-07-03	12:15:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-04-02	13:55:00
2015-07-02	12:15:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2015-12-31	13:55:00
2016-11-25	12:15:00
2016-12-23	12:15:00
2017-07-03	12:15:00
2017-11-24	12:15:00
2017-12-22	12:15:00
2018-07-03	12:15:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-07-03	12:15:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-07-02	12:15:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00
2022-11-25	12:05:00

Late Opens

The following table shows the late opens for the Live Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2011-11-24	17:00:00
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-17	09:05:00
2012-02-21	09:05:00
2012-05-29	09:05:00
2012-07-05	09:05:00
2012-09-04	09:05:00
2012-11-22	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-22	09:05:00
2013-02-19	09:05:00
2013-05-28	09:05:00
2013-07-05	09:05:00
2013-09-03	09:05:00
2013-11-29	09:05:00
2013-12-26	09:05:00

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2014-01-02	09:05:00
2014-01-21	09:05:00
2014-02-18	09:05:00
2014-04-21	09:05:00
2014-05-27	09:05:00
2014-07-07	09:05:00
2014-09-02	09:05:00
2015-01-20	09:05:00
2015-02-17	09:05:00
2015-04-06	09:05:00
2015-05-26	09:05:00
2015-07-06	09:05:00
2015-09-08	09:05:00
2015-12-28	09:05:00
2016-01-04	09:05:00
2016-01-19	09:05:00
2016-02-16	09:05:00

Time Zone

The Live Cattle Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M2K

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Russell 2000 Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro E-mini Russell 2000 Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

M6A

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6B

Introduction

This page shows the trading hours, holidays, and time zone of the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6C

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/CAD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/CAD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/CAD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/CAD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro USD/CAD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro USD/CAD Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6E

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6J

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/JPY Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/JPY Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/JPY Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/JPY Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro USD/JPY Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro USD/JPY Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6S

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/CHF Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/CHF Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/CHF Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/CHF Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro USD/CHF Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro USD/CHF Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MBT

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Bitcoin Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Bitcoin Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MCD

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MES

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

MET

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Ether Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Ether Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MIR

Introduction

This page shows the trading hours, holidays, and time zone of the Micro INR/USD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro INR/USD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro INR/USD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro INR/USD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro INR/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro INR/USD Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MJY

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MNH

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/CNH Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/CNH Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/CNH Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/CNH Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro USD/CNH Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro USD/CNH Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MNQ

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

MSF

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

NKD

Introduction

This page shows the trading hours, holidays, and time zone of the Nikkei/USD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Nikkei/USD Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Nikkei/USD Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Nikkei/USD Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Nikkei/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The Nikkei/USD Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

NQ

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Nasdaq-100 Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini Nasdaq-100 Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

RTY

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Russell 2000 Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The E-mini Russell 2000 Index Futures contract in the CME Future market trades in the `America/New York` time zone.

Market Hours

COMEX

Introduction

This page shows the trading hours, holidays, and time zone of the COMEX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COMEX Future market:

Weekday	Time (America/Chicago)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COMEX Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall COMEX Future market:

Symbol	Name
AUP	Aluminum MW U.S. Transaction Premium Platts (25MT) Futures
EDP	Aluminium European Premium Duty-Paid (Metal Bulletin) Futures
GC	Gold Futures
HG	Copper Futures
MGC	Micro Gold Futures
MGT	Micro Gold TAS Futures
SI	Silver Futures
SIL	Micro Silver Futures

COMEX

AUP

Introduction

This page shows the trading hours, holidays, and time zone of the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

EDP

Introduction

This page shows the trading hours, holidays, and time zone of the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

GC

Introduction

This page shows the trading hours, holidays, and time zone of the Gold Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

HG

Introduction

This page shows the trading hours, holidays, and time zone of the Copper Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Copper Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Copper Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Copper Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Copper Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Copper Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

MGC

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Gold Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Gold Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

MGT

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Gold TAS Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Gold TAS Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Gold TAS Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Gold TAS Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Gold TAS Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00
2022-11-24	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Gold TAS Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

SI

Introduction

This page shows the trading hours, holidays, and time zone of the Silver Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

SIL

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Silver Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Silver Futures contract in the COMEX Future market trades in the `America/New York` time zone.

Market Hours

ICE

Introduction

This page shows the trading hours, holidays, and time zone of the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The ICE Future market trades in the `America/New York` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall ICE Future market:

Symbol	Name
B	Brent Crude Futures
CC	Cocoa Futures
CT	Cotton No. 2 Futures
DX	US Dollar Index Futures
G	Low Sulfur Gasoil
KC	Coffee C Arabica Futures
OJ	Frozen Concentrated Orange Juice
SB	Sugar No. 11 Futures

ICE

B

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Crude Futures contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Brent Crude Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	20:00:00 to 24:00:00
Monday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Brent Crude Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:30:00
2013-02-18	13:30:00
2013-05-27	13:30:00
2013-07-04	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-09-02	13:30:00
2013-11-28	13:30:00
2013-11-29	14:00:00
2013-12-24	14:00:00
2013-12-31	15:00:00
2014-01-20	13:30:00
2014-02-17	13:30:00
2014-05-26	13:30:00
2014-07-04	13:30:00
2014-09-01	13:30:00
2014-11-27	13:30:00
2014-11-28	14:00:00
2014-12-24	14:00:00
2014-12-31	15:00:00
2015-01-19	13:30:00
2015-02-16	13:30:00
2015-06-03	13:30:00
2015-07-03	13:30:00
2015-09-07	13:30:00
2015-11-26	13:30:00
2015-11-27	14:00:00
2015-12-24	14:00:00
2015-12-31	15:00:00
2016-01-18	13:30:00
2016-02-15	13:30:00
2016-05-30	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:30:00
2016-09-05	13:30:00
2016-11-26	13:30:00
2016-11-27	14:00:00
2016-12-24	14:00:00
2016-12-30	15:00:00
2017-09-04	13:30:00
2017-11-23	13:30:00
2017-11-24	14:00:00
2017-12-22	17:00:00
2017-12-29	17:00:00
2018-07-04	13:30:00
2018-09-03	13:30:00
2018-11-22	13:30:00
2018-11-23	14:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-05-27	13:30:00
2019-07-04	13:30:00
2019-09-02	13:30:00
2019-11-28	13:30:00
2019-11-29	14:00:00
2019-12-24	14:00:00
2019-12-31	15:00:00
2020-05-25	13:30:00
2020-07-03	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-09-07	13:30:00
2020-11-26	13:30:00
2020-11-27	14:00:00
2020-12-24	14:00:00
2020-12-31	15:00:00

Late Opens

There are no days with late opens.

Time Zone

The Brent Crude Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

CC

Introduction

This page shows the trading hours, holidays, and time zone of the Cocoa Futures contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Cocoa Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	04:45:00 to 13:30:00
Tuesday	04:45:00 to 13:30:00
Wednesday	04:45:00 to 13:30:00
Thursday	04:45:00 to 13:30:00
Friday	04:45:00 to 13:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Cocoa Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-12-24	13:05:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Cocoa Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-04-09	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-04-01	07:30:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-04-21	07:30:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-04-06	07:30:00
2015-12-28	08:00:00
2016-12-27	08:00:00
2017-04-17	07:30:00
2017-12-26	08:00:00
2018-04-02	07:30:00
2018-12-26	08:00:00
2019-04-22	07:30:00
2019-12-26	08:00:00

Time Zone

The Cocoa Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

CT

Introduction

This page shows the trading hours, holidays, and time zone of the Cotton No. 2 Futures contract in the ICE Future market.

Pre-market Hours

The following table shows the pre-market hours for the Cotton No. 2 Futures contract in the ICE Future market:

Weekday	Time (America/New York)
Sunday	19:30:00 to 21:00:00
Monday	19:30:00 to 21:00:00
Tuesday	19:30:00 to 21:00:00
Wednesday	19:30:00 to 21:00:00
Thursday	19:30:00 to 21:00:00

Regular Trading Hours

The following table shows the regular trading hours for the Cotton No. 2 Futures contract in the ICE Future market:

Weekday	Time (America/New York)
Sunday	21:00:00 to 24:00:00
Monday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Tuesday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Wednesday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Thursday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Friday	00:00:00 to 14:20:00

Post-market Hours

The following table shows the post-market hours for the Cotton No. 2 Futures contract in the ICE Future market:

Weekday	Time (America/New York)
Monday	14:50:00 to 18:00:00
Tuesday	14:50:00 to 18:00:00
Wednesday	14:50:00 to 18:00:00
Thursday	14:50:00 to 18:00:00
Friday	14:50:00 to 18:00:00

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31

Date (*yyyy-mm-dd*)				
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Cotton No. 2 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-11-23	13:30:00
2017-11-24	13:30:00
2017-12-24	13:05:00
2019-11-29	13:30:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Cotton No. 2 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-07-05	08:00:00
2013-11-29	08:00:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-07-07	08:00:00
2014-11-28	08:00:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-11-27	08:00:00
2015-12-28	08:00:00
2016-11-25	08:00:00
2017-11-24	08:00:00
2017-12-26	08:00:00
2018-11-23	08:00:00
2018-12-26	08:00:00
2019-11-29	08:00:00
2019-12-26	08:00:00

Time Zone

The Cotton No. 2 Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

DX

Introduction

This page shows the trading hours, holidays, and time zone of the US Dollar Index Futures contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Dollar Index Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21

Date (*yyyy-mm-dd*)				
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25

Date (*yyyy-mm-dd*)				
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the US Dollar Index Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2007-12-24	13:15:00
2007-12-31	16:15:00
2008-01-21	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-02-18	13:00:00
2008-03-20	16:15:00
2008-07-03	16:15:00
2008-07-04	13:00:00
2008-08-29	16:15:00
2008-09-01	13:00:00
2008-11-26	17:00:00
2008-11-27	13:00:00
2008-11-28	13:15:00
2008-12-24	13:15:00
2008-12-26	17:00:00
2008-12-31	17:00:00
2009-01-02	17:00:00
2009-01-16	16:15:00
2009-01-19	13:00:00
2009-02-13	16:15:00
2009-02-16	13:00:00
2009-04-09	16:15:00
2009-05-22	16:15:00
2009-05-25	13:00:00
2009-07-02	16:15:00
2009-07-03	13:00:00
2009-09-04	16:15:00
2009-09-07	13:00:00
2009-10-09	16:15:00
2009-11-25	17:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-11-26	13:00:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2009-12-31	17:00:00
2010-01-15	16:15:00
2010-01-18	13:00:00
2010-02-12	16:15:00
2010-02-15	13:00:00
2010-04-02	11:15:00
2010-05-28	16:15:00
2010-05-31	13:00:00
2010-07-02	16:15:00
2010-07-05	13:00:00
2010-09-03	16:15:00
2010-09-06	13:00:00
2010-10-08	16:15:00
2010-11-25	13:00:00
2010-11-26	13:15:00
2010-12-23	17:00:00
2010-12-31	13:15:00
2011-01-14	16:15:00
2011-01-17	13:00:00
2011-02-18	16:15:00
2011-02-21	13:00:00
2011-05-27	16:15:00
2011-05-30	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2011-07-01	16:15:00
2011-07-04	13:00:00
2011-09-02	16:15:00
2011-09-05	13:00:00
2011-10-07	16:15:00
2011-11-24	13:00:00
2011-11-25	13:15:00
2012-02-17	16:15:00
2012-02-20	13:00:00
2012-04-06	11:15:00
2012-05-25	16:15:00
2012-05-28	13:00:00
2012-07-04	13:00:00
2012-08-31	16:15:00
2012-09-03	13:00:00
2012-10-05	16:15:00
2012-11-22	13:00:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-02-15	16:15:00
2013-02-18	13:00:00
2013-05-24	16:15:00
2013-05-27	13:00:00
2013-07-04	13:00:00
2013-08-30	16:15:00
2013-09-02	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-11-28	13:00:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-17	16:15:00
2014-01-20	13:00:00
2014-02-14	16:15:00
2014-02-17	13:00:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	11:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00

Late Opens

The following table shows the late opens for the US Dollar Index Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2008-12-26	06:00:00
2009-01-02	06:00:00
2011-12-27	06:00:00
2012-01-03	06:00:00
2012-12-26	06:00:00
2013-01-02	06:00:00
2013-12-26	06:00:00
2014-01-02	06:00:00

Time Zone

The US Dollar Index Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

G

Introduction

This page shows the trading hours, holidays, and time zone of the Low Sulfur Gasoil contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Low Sulfur Gasoil contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	20:00:00 to 24:00:00
Monday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Low Sulfur Gasoil contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:30:00
2013-02-18	13:30:00
2013-05-27	13:30:00
2013-07-04	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-09-02	13:30:00
2013-11-28	13:30:00
2013-11-29	14:00:00
2013-12-24	14:00:00
2013-12-31	15:00:00
2014-01-20	13:30:00
2014-02-17	13:30:00
2014-05-26	13:30:00
2014-07-04	13:30:00
2014-09-01	13:30:00
2014-11-27	13:30:00
2014-11-28	14:00:00
2014-12-24	14:00:00
2014-12-31	15:00:00
2015-01-19	13:30:00
2015-02-16	13:30:00
2015-06-03	13:30:00
2015-07-03	13:30:00
2015-09-07	13:30:00
2015-11-26	13:30:00
2015-11-27	14:00:00
2015-12-24	14:00:00
2015-12-31	15:00:00
2016-01-18	13:30:00
2016-02-15	13:30:00
2016-05-30	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:30:00
2016-09-05	13:30:00
2016-11-26	13:30:00
2016-11-27	14:00:00
2016-12-24	14:00:00
2016-12-30	15:00:00
2017-09-04	13:30:00
2017-11-23	13:30:00
2017-11-24	14:00:00
2017-12-22	17:00:00
2017-12-29	17:00:00
2018-07-04	13:30:00
2018-09-03	13:30:00
2018-11-22	13:30:00
2018-11-23	14:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-05-27	13:30:00
2019-07-04	13:30:00
2019-09-02	13:30:00
2019-11-28	13:30:00
2019-11-29	14:00:00
2019-12-24	14:00:00
2019-12-31	15:00:00
2020-05-25	13:30:00
2020-07-03	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2020-09-07	13:30:00
2020-11-26	13:30:00
2020-11-27	14:00:00
2020-12-24	14:00:00
2020-12-31	15:00:00

Late Opens

There are no days with late opens.

Time Zone

The Low Sulfur Gasoil contract in the ICE Future market trades in the `America/New York` time zone.

ICE

KC

Introduction

This page shows the trading hours, holidays, and time zone of the Coffee C Arabica Futures contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Coffee C Arabica Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	04:15:00 to 13:30:00
Tuesday	04:15:00 to 13:30:00
Wednesday	04:15:00 to 13:30:00
Thursday	04:15:00 to 13:30:00
Friday	04:15:00 to 13:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Coffee C Arabica Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-12-24	13:05:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Coffee C Arabica Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-04-09	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-04-01	07:30:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-04-21	07:30:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-04-06	07:30:00
2015-12-28	08:00:00
2016-12-27	08:00:00
2017-04-17	07:30:00
2017-12-26	08:00:00
2018-04-02	07:30:00
2018-12-26	08:00:00
2019-04-22	07:30:00
2019-12-26	08:00:00

Time Zone

The Coffee C Arabica Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

OJ

Introduction

This page shows the trading hours, holidays, and time zone of the Frozen Concentrated Orange Juice contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	08:00:00 to 14:00:00
Tuesday	08:00:00 to 14:00:00
Wednesday	08:00:00 to 14:00:00
Thursday	08:00:00 to 14:00:00
Friday	08:00:00 to 14:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-11-23	13:30:00
2017-11-24	13:30:00
2017-12-24	13:05:00
2019-11-29	13:30:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2012-12-26	07:30:00

Time Zone

The Frozen Concentrated Orange Juice contract in the ICE Future market trades in the `America/New York` time zone.

ICE

SB

Introduction

This page shows the trading hours, holidays, and time zone of the Sugar No. 11 Futures contract in the ICE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Sugar No. 11 Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	03:30:00 to 13:00:00
Tuesday	03:30:00 to 13:00:00
Wednesday	03:30:00 to 13:00:00
Thursday	03:30:00 to 13:00:00
Friday	03:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Sugar No. 11 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00

Late Opens

The following table shows the late opens for the Sugar No. 11 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-04-09	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-04-01	07:30:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-04-21	07:30:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-04-06	07:30:00
2015-12-28	08:00:00
2016-12-27	08:00:00
2017-04-17	07:30:00
2017-12-26	08:00:00
2018-04-02	07:30:00
2018-12-26	08:00:00
2019-04-22	07:30:00
2019-12-26	08:00:00

Time Zone

The Sugar No. 11 Futures contract in the ICE Future market trades in the `America/New York` time zone.

Market Hours

INDIA

Introduction

This page shows the trading hours, holidays, and time zone of the INDIA Future market.

Pre-market Hours

The following table shows the pre-market hours for the INDIA Future market:

Weekday	Time (Asia/Kolkata)
Monday	09:00:00 to 09:15:00
Tuesday	09:00:00 to 09:15:00
Wednesday	09:00:00 to 09:15:00
Thursday	09:00:00 to 09:15:00
Friday	09:00:00 to 09:15:00

Regular Trading Hours

The following table shows the regular trading hours for the INDIA Future market:

Weekday	Time (Asia/Kolkata)
Monday	09:15:00 to 15:30:00
Tuesday	09:15:00 to 15:30:00
Wednesday	09:15:00 to 15:30:00
Thursday	09:15:00 to 15:30:00
Friday	09:15:00 to 15:30:00

Post-market Hours

The following table shows the post-market hours for the INDIA Future market:

Weekday	Time (Asia/Kolkata)
Monday	15:40:00 to 16:00:00
Tuesday	15:40:00 to 16:00:00
Wednesday	15:40:00 to 16:00:00
Thursday	15:40:00 to 16:00:00
Friday	15:40:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the INDIA Future market:

Date (*yyyy-mm-dd*)				
2004-01-26	2004-04-14	2005-01-26	2005-04-14	2005-08-15
2006-01-26	2006-04-14	2006-05-01	2006-08-15	2006-10-02
2006-12-25	2007-01-26	2007-05-01	2007-08-15	2007-10-02
2007-12-25	2008-04-14	2008-05-01	2008-08-15	2008-10-02
2008-12-25	2009-01-26	2009-04-14	2009-05-01	2009-10-02
2009-12-25	2010-01-26	2010-04-14	2011-01-26	2011-03-02
2011-04-12	2011-04-14	2011-04-22	2011-08-15	2011-08-31
2011-09-01	2011-10-06	2011-10-26	2011-10-27	2011-11-07
2011-11-10	2011-12-06	2012-01-26	2012-02-20	2012-03-08
2012-04-05	2012-04-06	2012-05-01	2012-08-15	2012-08-20
2012-09-19	2012-10-02	2012-10-24	2012-11-14	2012-11-28
2012-12-25	2013-03-27	2013-03-29	2013-04-19	2013-04-24
2013-05-01	2013-08-09	2013-08-15	2013-09-09	2013-10-02
2013-10-16	2013-11-04	2013-11-15	2013-12-25	2014-02-27
2014-03-17	2014-04-08	2014-04-14	2014-04-18	2014-04-24
2014-05-01	2014-07-29	2014-08-15	2014-08-29	2014-10-02
2014-10-03	2014-10-06	2014-10-15	2014-10-24	2014-11-04
2014-11-06	2014-12-25	2015-01-26	2015-02-17	2015-03-06

Date (*yyyy-mm-dd*)				
2015-04-02	2015-04-03	2015-04-14	2015-05-01	2015-09-17
2015-09-25	2015-10-02	2015-10-22	2015-11-12	2015-11-25
2015-12-25	2016-01-26	2016-03-07	2016-03-24	2016-03-25
2016-04-14	2016-04-15	2016-04-19	2016-07-06	2016-08-15
2016-09-05	2016-09-13	2016-10-11	2016-10-12	2016-10-31
2016-11-14	2017-01-26	2017-02-24	2017-03-13	2017-04-04
2017-04-14	2017-05-01	2017-06-26	2017-08-15	2017-08-25
2017-10-02	2017-10-20	2017-12-25	2018-01-26	2018-02-13
2018-03-02	2018-03-29	2018-03-30	2018-05-01	2018-08-15
2018-08-22	2018-09-13	2018-09-20	2018-10-02	2018-10-18
2018-11-08	2018-11-23	2018-12-25	2019-03-04	2019-03-21
2019-04-17	2019-04-19	2019-04-29	2019-05-01	2019-06-05
2019-08-12	2019-08-15	2019-09-02	2019-09-10	2019-10-02
2019-10-08	2019-10-21	2019-10-28	2019-11-12	2019-12-25
2020-02-21	2020-03-10	2020-04-02	2020-04-06	2020-04-10
2020-04-14	2020-05-01	2020-05-25	2020-10-02	2020-11-16
2020-11-30	2020-12-25	2021-01-26	2021-03-11	2021-03-29
2021-04-02	2021-04-14	2021-04-21	2021-05-13	2021-07-21
2021-08-19	2021-09-10	2021-10-15	2021-11-05	2021-11-19
2022-01-26	2022-04-14	2022-08-15	2023-01-26	2023-04-14
2023-05-01	2023-08-15	2023-10-02	2023-12-25	2024-01-26
2024-05-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The INDIA Future market trades in the `Asia/Kolkata` time zone.

Market Hours

NFO

Introduction

This page shows the trading hours, holidays, and time zone of the NFO Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NFO Future market:

Weekday	**Time (Asia/Kolkata)**
Monday	09:15:00 to 15:30:00
Tuesday	09:15:00 to 15:30:00
Wednesday	09:15:00 to 15:30:00
Thursday	09:15:00 to 15:30:00
Friday	09:15:00 to 15:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NFO Future market:

Date (*yyyy-mm-dd*)				
2004-01-26	2004-04-14	2005-01-26	2005-04-14	2005-08-15
2006-01-26	2006-04-14	2006-05-01	2006-08-15	2006-10-02
2006-12-25	2007-01-26	2007-05-01	2007-08-15	2007-10-02
2007-12-25	2008-04-14	2008-05-01	2008-08-15	2008-10-02
2008-12-25	2009-01-26	2009-04-14	2009-05-01	2009-10-02
2009-12-25	2010-01-26	2010-04-14	2011-01-26	2011-03-02

Date (*yyyy-mm-dd*)				
2011-04-12	2011-04-14	2011-04-22	2011-08-15	2011-08-31
2011-09-01	2011-10-06	2011-10-26	2011-10-27	2011-11-07
2011-11-10	2011-12-06	2012-01-26	2012-02-20	2012-03-08
2012-04-05	2012-04-06	2012-05-01	2012-08-15	2012-08-20
2012-09-19	2012-10-02	2012-10-24	2012-11-14	2012-11-28
2012-12-25	2013-03-27	2013-03-29	2013-04-19	2013-04-24
2013-05-01	2013-08-09	2013-08-15	2013-09-09	2013-10-02
2013-10-16	2013-11-04	2013-11-15	2013-12-25	2014-02-27
2014-03-17	2014-04-08	2014-04-14	2014-04-18	2014-04-24
2014-05-01	2014-07-29	2014-08-15	2014-08-29	2014-10-02
2014-10-03	2014-10-06	2014-10-15	2014-10-24	2014-11-04
2014-11-06	2014-12-25	2015-01-26	2015-02-17	2015-03-06
2015-04-02	2015-04-03	2015-04-14	2015-05-01	2015-09-17
2015-09-25	2015-10-02	2015-10-22	2015-11-12	2015-11-25
2015-12-25	2016-01-26	2016-03-07	2016-03-24	2016-03-25
2016-04-14	2016-04-15	2016-04-19	2016-07-06	2016-08-15
2016-09-05	2016-09-13	2016-10-11	2016-10-12	2016-10-31
2016-11-14	2017-01-26	2017-02-24	2017-03-13	2017-04-04
2017-04-14	2017-05-01	2017-06-26	2017-08-15	2017-08-25
2017-10-02	2017-10-20	2017-12-25	2018-01-26	2018-02-13
2018-03-02	2018-03-29	2018-03-30	2018-05-01	2018-08-15
2018-08-22	2018-09-13	2018-09-20	2018-10-02	2018-10-18
2018-11-08	2018-11-23	2018-12-25	2019-03-04	2019-03-21
2019-04-17	2019-04-19	2019-04-29	2019-05-01	2019-06-05
2019-08-12	2019-08-15	2019-09-02	2019-09-10	2019-10-02
2019-10-08	2019-10-21	2019-10-28	2019-11-12	2019-12-25

Date (*yyyy-mm-dd*)				
2020-02-21	2020-03-10	2020-04-02	2020-04-06	2020-04-10
2020-04-14	2020-05-01	2020-05-25	2020-10-02	2020-11-16
2020-11-30	2020-12-25	2021-01-26	2021-03-11	2021-03-29
2021-04-02	2021-04-14	2021-04-21	2021-05-13	2021-07-21
2021-08-19	2021-09-10	2021-10-15	2021-11-05	2021-11-19
2022-01-26	2022-04-14	2022-08-15	2023-01-26	2023-04-14
2023-05-01	2023-08-15	2023-10-02	2023-12-25	2024-01-26
2024-05-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NFO Future market trades in the `Asia/Kolkata` time zone.

Market Hours

NYMEX

Introduction

This page shows the trading hours, holidays, and time zone of the NYMEX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NYMEX Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall NYMEX Future market:

Symbol	Name
1S	Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures
22	Argus Propane Far East Index BALMO Futures
A0D	Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
A0F	Mini Singapore Fuel Oil 180 cst (Platts) Futures
A1L	Gulf Coast ULSD (Platts) Up-Down BALMO Futures
A1M	Gulf Coast Jet (Platts) Up-Down BALMO Futures
A1R	Propane Non-LDH Mont Belvieu (OPIS) Futures
A32	European Propane CIF ARA (Argus) BALMO Futures
A3G	Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures
A7E	Argus Propane Far East Index Futures
A7I	Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures
A7Q	Mont Belvieu Natural Gasoline (OPIS) Futures
A8J	Mont Belvieu Normal Butane (OPIS) BALMO Futures
A8K	Conway Propane (OPIS) Futures
A8O	Mont Belvieu LDH Propane (OPIS) BALMO Futures
A91	Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures
A9N	Argus Propane (Saudi Aramco) Futures
AA6	Group Three ULSD (Platts) vs. NY Harbor ULSD Futures
AA8	Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures
ABS	Singapore Fuel Oil 180 cst (Platts) BALMO Futures
ABT	Singapore Fuel Oil 380 cst (Platts) BALMO Futures
AC0	Mont Belvieu Ethane (OPIS) Futures

Symbol	Name
AD0	Mont Belvieu Normal Butane (OPIS) Futures
ADB	Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures
AE5	Argus LLS vs. WTI (Argus) Trade Month Futures
AGA	Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures
AJL	Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures
AJS	Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures
AKL	Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures
AKZ	European Naphtha (Platts) BALMO Futures
APS	European Propane CIF ARA (Argus) Futures
AR0	Mont Belvieu Natural Gasoline (OPIS) BALMO Futures
ARE	RBOB Gasoline Crack Spread Futures
AVZ	Gulf Coast HSFO (Platts) BALMO Futures
AYV	Mars (Argus) vs. WTI Trade Month Futures
AYX	Mars (Argus) vs. WTI Financial Futures
AZ1	Ethanol T2 FOB Rdam Including Duty (Platts) Futures
B0	Mont Belvieu LDH Propane (OPIS) Futures
B7H	Gasoline Euro-bob Oxy NWE Barges (Argus) Futures
BK	WTI-Brent Financial Futures
BOO	3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures
BR7	Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures
BZ	Brent Last Day Financial Futures
CL	Crude Oil Futures
CRB	Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures
CSW	Clearbrook Bakken Sweet (NE2) Monthly Index Futures

Symbol	Name
CSX	WTI Financial Futures
CU	Chicago Ethanol (Platts) Futures
D1N	Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures
DCB	Dubai Crude Oil (Platts) Financial Futures
E6	Japan C&F Naphtha (Platts) BALMO Futures
EN	European Naphtha (Platts) Crack Spread Futures
EPN	European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures
EVC	Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
EWG	East-West Gasoline Spread (Platts-Argus) Futures
EWN	East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures
EXR	RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures
FO	3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures
FRC	Freight Route TC14 (Baltic) Futures
FSS	1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
GCU	Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
HCL	WTI Houston Crude Oil Futures
HH	Natural Gas (Henry Hub) Last-day Financial Futures
HO	NY Harbor ULSD Futures
HP	Natural Gas (Henry Hub) Penultimate Financial Futures
HRC	U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures
HTT	WTI Houston (Argus) vs. WTI Trade Month Futures
M1B	Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures

Symbol	Name
M35	Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures
M5F	Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures
MAF	Micro Singapore Fuel Oil 380CST (Platts) Futures
MCL	Micro WTI Crude Oil Futures
MEF	Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
NG	Henry Hub Natural Gas Futures
PA	Palladium Futures
PAM	Micro Palladium Futures
PL	Platinum Futures
R5O	Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures
RB	RBOB Gasoline Futures
S5O	Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures
YO	No. 11 Sugar Futures

NYMEX

1S

Introduction

This page shows the trading hours, holidays, and time zone of the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

22

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A0D

Introduction

This page shows the trading hours, holidays, and time zone of the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A0F

Introduction

This page shows the trading hours, holidays, and time zone of the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A1L

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A1M

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A1R

Introduction

This page shows the trading hours, holidays, and time zone of the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A32

Introduction

This page shows the trading hours, holidays, and time zone of the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A3G

Introduction

This page shows the trading hours, holidays, and time zone of the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A7E

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane Far East Index Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Argus Propane Far East Index Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A7I

Introduction

This page shows the trading hours, holidays, and time zone of the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A7Q

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A8J

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A8K

Introduction

This page shows the trading hours, holidays, and time zone of the Conway Propane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Conway Propane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A8O

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A91

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A9N

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AA6

Introduction

This page shows the trading hours, holidays, and time zone of the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AA8

Introduction

This page shows the trading hours, holidays, and time zone of the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ABS

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ABT

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AC0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AD0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ADB

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AE5

Introduction

This page shows the trading hours, holidays, and time zone of the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AGA

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AJL

Introduction

This page shows the trading hours, holidays, and time zone of the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AJS

Introduction

This page shows the trading hours, holidays, and time zone of the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AKL

Introduction

This page shows the trading hours, holidays, and time zone of the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AKZ

Introduction

This page shows the trading hours, holidays, and time zone of the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

APS

Introduction

This page shows the trading hours, holidays, and time zone of the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AR0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ARE

Introduction

This page shows the trading hours, holidays, and time zone of the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AVZ

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AYV

Introduction

This page shows the trading hours, holidays, and time zone of the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AYX

Introduction

This page shows the trading hours, holidays, and time zone of the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AZ1

Introduction

This page shows the trading hours, holidays, and time zone of the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

B0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

B7H

Introduction

This page shows the trading hours, holidays, and time zone of the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BK

Introduction

This page shows the trading hours, holidays, and time zone of the WTI-Brent Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The WTI-Brent Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BOO

Introduction

This page shows the trading hours, holidays, and time zone of the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BR7

Introduction

This page shows the trading hours, holidays, and time zone of the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BZ

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Last Day Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Brent Last Day Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CL

Introduction

This page shows the trading hours, holidays, and time zone of the Crude Oil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Crude Oil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CRB

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CSW

Introduction

This page shows the trading hours, holidays, and time zone of the Clearbrook Bakken Sweet (NE2) Monthly Index Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Clearbrook Bakken Sweet (NE2) Monthly Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Clearbrook Bakken Sweet (NE2) Monthly Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Clearbrook Bakken Sweet (NE2) Monthly Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Clearbrook Bakken Sweet (NE2) Monthly Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Clearbrook Bakken Sweet (NE2) Monthly Index Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CSX

Introduction

This page shows the trading hours, holidays, and time zone of the WTI Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The WTI Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CU

Introduction

This page shows the trading hours, holidays, and time zone of the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Chicago Ethanol (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

D1N

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

DCB

Introduction

This page shows the trading hours, holidays, and time zone of the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

E6

Introduction

This page shows the trading hours, holidays, and time zone of the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EN

Introduction

This page shows the trading hours, holidays, and time zone of the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EPN

Introduction

This page shows the trading hours, holidays, and time zone of the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EVC

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EWG

Introduction

This page shows the trading hours, holidays, and time zone of the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EWN

Introduction

This page shows the trading hours, holidays, and time zone of the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EXR

Introduction

This page shows the trading hours, holidays, and time zone of the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

FO

Introduction

This page shows the trading hours, holidays, and time zone of the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

FRC

Introduction

This page shows the trading hours, holidays, and time zone of the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

FSS

Introduction

This page shows the trading hours, holidays, and time zone of the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

GCU

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HCL

Introduction

This page shows the trading hours, holidays, and time zone of the WTI Houston Crude Oil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The WTI Houston Crude Oil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HH

Introduction

This page shows the trading hours, holidays, and time zone of the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HO

Introduction

This page shows the trading hours, holidays, and time zone of the NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HP

Introduction

This page shows the trading hours, holidays, and time zone of the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HRC

Introduction

This page shows the trading hours, holidays, and time zone of the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HTT

Introduction

This page shows the trading hours, holidays, and time zone of the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

M1B

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

M35

Introduction

This page shows the trading hours, holidays, and time zone of the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

M5F

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

MAF

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

MCL

Introduction

This page shows the trading hours, holidays, and time zone of the Micro WTI Crude Oil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00
2022-11-24	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro WTI Crude Oil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

MEF

Introduction

This page shows the trading hours, holidays, and time zone of the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

NG

Introduction

This page shows the trading hours, holidays, and time zone of the Henry Hub Natural Gas Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Henry Hub Natural Gas Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

PA

Introduction

This page shows the trading hours, holidays, and time zone of the Palladium Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Palladium Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

PAM

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Palladium Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00
2022-11-24	13:30:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Palladium Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

PL

Introduction

This page shows the trading hours, holidays, and time zone of the Platinum Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Platinum Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Platinum Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Platinum Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Platinum Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

R5O

Introduction

This page shows the trading hours, holidays, and time zone of the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

RB

Introduction

This page shows the trading hours, holidays, and time zone of the RBOB Gasoline Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The RBOB Gasoline Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

S5O

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25

Date (*yyyy-mm-dd*)				
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

The following table shows the early closes for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

YO

Introduction

This page shows the trading hours, holidays, and time zone of the No. 11 Sugar Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the No. 11 Sugar Futures contract in the NYMEX Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the No. 11 Sugar Futures contract in the NYMEX Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the No. 11 Sugar Futures contract in the NYMEX Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

Date (*yyyy-mm-dd*)				
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01

Date (*yyyy-mm-dd*)				
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The No. 11 Sugar Futures contract in the NYMEX Future market trades in the `America/Chicago` time zone.

Market Hours

NYSELIFFE

Introduction

This page shows the trading hours, holidays, and time zone of the NYSELIFFE Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NYSELIFFE Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NYSELIFFE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

Date (*yyyy-mm-dd*)				
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26

Date (*yyyy-mm-dd*)				
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NYSELIFFE Future market trades in the `America/New York` time zone.

Market Hours

SGX

Introduction

This page shows the trading hours, holidays, and time zone of the SGX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGX Future market:

Weekday	Time (Asia/Singapore)
Monday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Tuesday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Wednesday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Thursday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Friday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	

Early Closes

The following table shows the early closes for the SGX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Singapore)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-04-02	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The SGX Future market trades in the `Asia/Singapore` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall SGX Future market:

Symbol	Name
IN	Nifty Indices Futures
NK	SGX Nikkei 225 Index Futures
TW	MSCI Taiwan Index Futures

SGX

IN

Introduction

This page shows the trading hours, holidays, and time zone of the Nifty Indices Futures contract in the SGX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nifty Indices Futures contract in the SGX Future market:

Weekday	Time (Asia/Singapore)
Monday	08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Tuesday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Wednesday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Thursday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Friday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Saturday	00:00:00 to 06:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Nifty Indices Futures contract in the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	

Early Closes

The following table shows the early closes for the Nifty Indices Futures contract in the SGX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Singapore)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-04-02	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The Nifty Indices Futures contract in the SGX Future market trades in the `Asia/Singapore` time zone.

SGX

NK

Introduction

This page shows the trading hours, holidays, and time zone of the SGX Nikkei 225 Index Futures contract in the SGX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGX Nikkei 225 Index Futures contract in the SGX Future market:

Weekday	Time (Asia/Singapore)
Monday	07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Tuesday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Wednesday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Thursday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Friday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Saturday	00:00:00 to 05:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the SGX Nikkei 225 Index Futures contract in the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	

Early Closes

The following table shows the early closes for the SGX Nikkei 225 Index Futures contract in the SGX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Singapore)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-04-02	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The SGX Nikkei 225 Index Futures contract in the SGX Future market trades in the `Asia/Singapore` time zone.

SGX

TW

Introduction

This page shows the trading hours, holidays, and time zone of the MSCI Taiwan Index Futures contract in the SGX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MSCI Taiwan Index Futures contract in the SGX Future market:

Weekday	Time (Asia/Singapore)
Monday	08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Tuesday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Wednesday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Thursday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Friday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Saturday	00:00:00 to 05:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the MSCI Taiwan Index Futures contract in the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	

Early Closes

The following table shows the early closes for the MSCI Taiwan Index Futures contract in the SGX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Singapore)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-04-02	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The MSCI Taiwan Index Futures contract in the SGX Future market trades in the `Asia/Singapore` time zone.

Asset Classes

Future Options

Future Option contracts give the buyer a window of opportunity to buy or sell the underlying Future contract at a specific price.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateFutureOptionAlgorithm.py
BasicTemplateFutureOptionAlgorithm.cs

Future Options

Requesting Data

Introduction

Request Future Options data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Future Options dataset listing . To trade Future Options live, you can use our Future Options data feed or one of the brokerage data feeds . We currently only support American-style Options for Future Options.

Create Subscriptions

Before you can subscribe to a Future Option contract, you may configure the underlying volatility model and you must get the contract `Symbol` .

Configure the Underlying Futures Contract

In most cases, you should subscribe to the underlying Futures contract before you subscribe to a Futures Option contract. If you don't, LEAN automatically subscribes to the underlying Futures contract with the following settings:

Setting	Value
Fill forward	Same as the Option contract
Leverage	0
Extended Market Hours	Same as the Option contract

In this case, you still need the Futures contract `Symbol` to subscribe to Futures Option contracts. If you don't have access to it, create it.

```
self.future_contract_symbol = Symbol.CreateFuture(Futures.Indices.SP500EMini,
    Market.CME, datetime(2022, 6, 17))
```

PY

For more information about getting the `Symbol` of Futures contracts, see Create Subscriptions .

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Get Contract Symbols

To subscribe to a Future Option contract, you need the contract `Symbol` . You can get the contract `Symbol` from the `CreateOption` method or from the `OptionChainProvider` . If you use the `CreateOption` method, you need to provide the contract details.

```
self.option_contract_symbol = Symbol.CreateOption(self.future_contract_symbol,
    Market.CME, OptionStyle.American, OptionRight.Call, 3600, datetime(2022, 6, 17))
```

Another way to get a Future Option contract `Symbol` is to use the `OptionChainProvider` . The `GetOptionContractList` method of `OptionChainProvider` returns a list of `Symbol` objects that reference the available Option contracts for a given underlying Future contract on a given date. The `Symbol` you pass to the method can reference any of the following Futures contracts:

- The continuous Futures contract
- A contract in the Futures universe
- A contract that you added with `AddFutureContract`

To filter and select contracts that the `GetOptionContractList` method returns, you can use the following properties of each `Symbol` object:

Property	Description
`ID.Date`	The expiration date of the contract.
`ID.StrikePrice`	The strike price of the contract.
`ID.OptionRight`	The contract type. The `OptionRight` enumeration has the following members:
`ID.OptionStyle`	The contract style. The `OptionStyle` enumeration has the following members: We currently only support American-style Options for Future Options.

```
option_contract_symbols = self.OptionChainProvider.GetOptionContractList(self.future_contract_symbol,
self.Time)
expiry = min([symbol.ID.Date for symbol in option_contract_symbols])
filtered_symbols = [symbol for symbol in option_contract_symbols if symbol.ID.Date == expiry and
symbol.ID.OptionRight == OptionRight.Call]
self.option_contract_symbol = sorted(filtered_symbols, key=lambda symbol: symbol.ID.StrikePrice)[0]
```

Subscribe to Contracts

To create a Future Option contract subscription, pass the contract `Symbol` to the `AddFutureOptionContract` method. Save a reference to the contract `Symbol` so you can easily access the Option contract in the OptionChain that LEAN passes to the `OnData` method. To override the default pricing model of the Option, set a pricing model .

```
option = self.AddFutureOptionContract(self.option_contract_symbol)
option.PriceModel = OptionPriceModels.BjerksundStensland()
```

The `AddFutureOptionContract` method creates a subscription for a single Option contract and adds it to your **user-defined** universe. To create a dynamic universe of Future Option contracts, add a Future Options universe .

Warm Up Contract Prices

If you subscribe to a Future Option contract with `AddFutureOptionContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
seeder = FuncSecuritySeeder(self.GetLastKnownPrices)
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel, seeder, self))
```

PY

Supported Assets

To view the supported assets in the US Future Options dataset, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Future Option contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

There is only one resolution option, so you don't need to pass a `resolution` argument to the `AddFutureOptionContract` method.

```
self.AddFutureOptionContract(self.option_contract_symbol, Resolution.Minute)
```

PY

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Future Options:

You don't need to pass a `Market` argument to the `AddFutureOptionContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.AddFutureOptionContract(self.option_contract_symbol, fillForward=False)
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Future Options are already leveraged products, so you can't change their leverage.

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
self.AddFutureOptionContract(self.option_contract_symbol, extendedMarketHours=True)
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Remove Subscriptions

To remove a contract subscription that you created with `AddFutureOptionContract` , call the `RemoveOptionContract` method. This method is an alias for `RemoveSecurity` .

```
self.RemoveOptionContract(self.option_contract_symbol)
```

The `RemoveOptionContract` method cancels your open orders for the contract and liquidates your holdings.

Properties

The `AddFutureOptionContract` method returns an `Option` object, which have the following properties:

Helper Methods

The `Option` object provides methods you can use for basic calculations. These methods require the underlying price. To get the `Option` object and the `Security` object for its underlying in any function, use the Option `Symbol` to access the value in the `Securities` object.

```
option = self.Securities[self.option_contract_symbol]
underlying = self.Securities[self.option_contract_symbol.Underlying]
underlying_price = underlying.Price
```

To get the Option payoff , call the `GetPayOff` method.

```
pay_off = option.GetPayOff(underlying_price)
```

To get the Option intrinsic value , call the `GetIntrinsicValue` method.

```
intrinsic_value = option.GetIntrinsicValue(underlying_price)
```
PY

To get the Option out-of-the-money amount , call the `OutOfTheMoneyAmount` method.

```
otm_amount = option.OutOfTheMoneyAmount(underlying_price)
```
PY

To check whether the Option can be automatic exercised, call the `IsAutoExercised` method.

```
is_auto_exercised = option.IsAutoExercised(underlying_price)
```
PY

Future Options

Handling Data

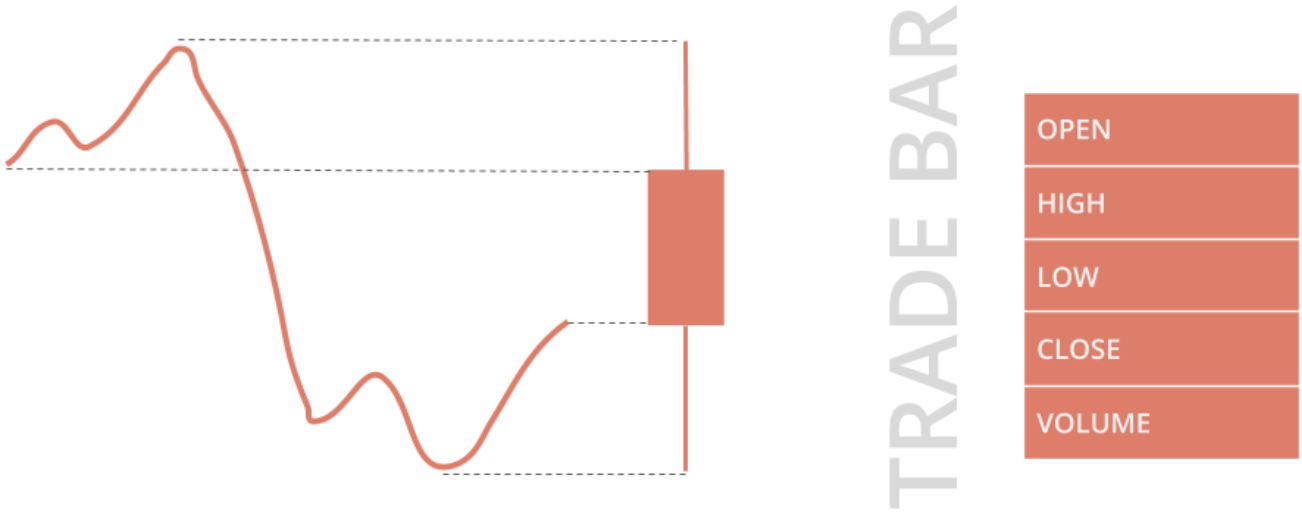
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the Futures Option contract ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the Option contract `Symbol` . If the Option contract doesn't actively trade or you are in the same time step as when you added the Option contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Option contract before you index the `Slice` with the Option contract `Symbol` .

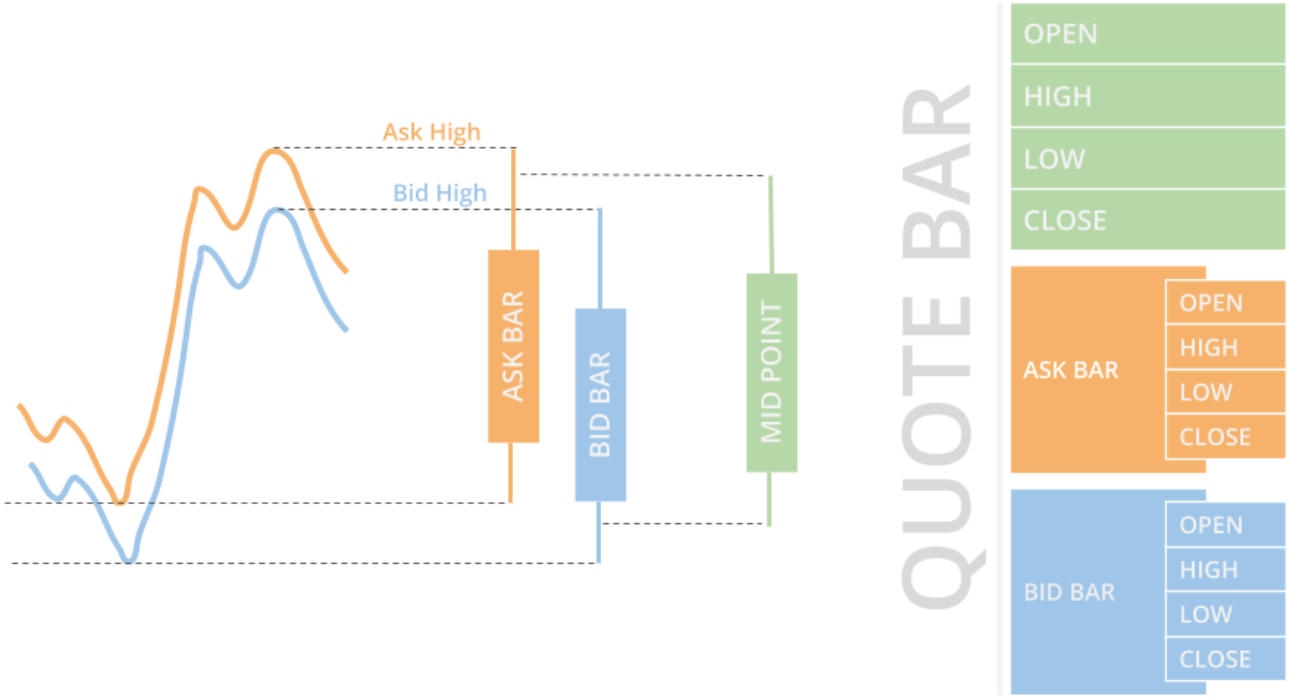
```
def OnData(self, slice: Slice) -> None:
    if self.option_contract_symbol in slice.Bars:
        trade_bar = slice.Bars[self.option_contract_symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the Option contract `Symbol` . If the Option contract doesn't actively get quotes or you are in the same time step as when you added the Option contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Option contract before you index the `Slice` with the Option contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.option_contract_symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.option_contract_symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Futures Chains

`FuturesChain` objects represent an entire chain of contracts for a single underlying Future. They have the following properties:

To get the `FuturesChain` , index the `FuturesChains` property of the `Slice` with the continuous contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.future_contract_symbol.Canonical)
    if chain:
        contracts = chain.Contracts
```

You can also loop through the `FuturesChains` property to get each `FuturesChain` .

```
def OnData(self, slice: Slice) -> None:
    for continuous_contract_symbol, chain in slice.FuturesChains.items():
        contracts = chain.Contracts
```

Futures Contracts

`FuturesContract` objects represent the data of a single Futures contract in the market. They have the following properties:

To get the Futures contracts in the `Slice` , use the `Contracts` property of the `FuturesChain` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.future_contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.future_contract_symbol)
        if contract:
            price = contract.LastPrice
```

Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security. They have the

following properties:

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.option_contract_symbol.Canonical)
    if chain:
        contracts = chain.Contracts
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    for canonical_fop_symbol, chain in slice.OptionChains.items():
        contracts = chain.Contracts
```

Option Contracts

`OptionContract` objects represent the data of a single Option contract in the market. They have the following properties:

To get the Option contracts in the `Slice` , use the `Contracts` property of the `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.option_contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.option_contract_symbol)
        if contract:
            price = contract.Price
```

You can also iterate through the `FuturesChains` first.

```
def OnData(self, slice: Slice) -> None:
    for continuous_future_symbol, futures_chain in slice.FuturesChains.items():
        # Select a Future Contract and create its canonical FOP Symbol
        futures_contract = [contract for contract in futures_chain][0]
        canonical_fop_symbol = Symbol.CreateCanonicalOption(futures_contract.Symbol)
        option_chain = slice.OptionChains.get(canonical_fop_symbol)
        if option_chain:
            option_contract = option_chain.Contracts.get(self.option_contract_symbol)
            if option_contract:
                price = option_contract.Price
```

Greeks and Implied Volatility

To get the Greeks and implied volatility of an Option contract, use the `Greeks` and `ImpliedVolatility` members.

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.option_contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.option_contract_symbol)
        if contract:
            delta = contract.Greeks.Delta
            iv = contract.ImpliedVolatility
```

LEAN only calculates Greeks and implied volatility when you request them because they are expensive operations. If you invoke the `Greeks` member, the Greeks aren't calculated. However, if you invoke the `Greeks.Delta` member, LEAN calculates the delta. To avoid unecessary computation in your algorithm, only request the Greeks and implied volatility when you need them. For more information about the Greeks and implied volatility, see Options Pricing .

Open Interest

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Option` or `OptionContract` .

PY

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            open_interest = contract.OpenInterest
```

Future Options

Market Hours

Introduction

The Future Option markets have the same hours as the Futures market. For more information, see the supported contracts and the market hours of Future markets .

Asset Classes

Index

There are cash Indices and volatility Indices. The former is a hypothetical portfolio that represents a segment of the financial market and the latter shows the expected level of price fluctuation in an Index Option.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateIndexAlgorithm.py
BasicTemplateIndexAlgorithm.cs

Index

Requesting Data

Introduction

Request Index data in your algorithm to receive a feed of Index prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Cash Indices dataset listing . To trade live with Index data, you can use one of the brokerage data feeds .

Create Subscriptions

To create an Index subscription, in the `Initialize` method, call the `AddIndex` method. The `AddIndex` method returns an `Index` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the Index data in the `Slice` .

```
self.symbol = self.AddIndex("VIX").Symbol
```

PY

To view the supported assets in the US Cash Indices dataset, see the Supported Indices .

Resolutions

The following table shows the available resolutions and data formats for Index subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	
`Second`	✓			
`Minute`	✓			
`Hour`	✓			
`Daily`	✓			

The default resolution for Index subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddIndex` method.

```
self.symbol = self.AddIndex("VIX", Resolution.Daily).Symbol
```

PY

To create custom resolution periods, see Consolidating Data .

Supported Markets

The only market available for Indices is `Market.USA` , so you don't need to pass a `market` argument to the `AddIndex` method.

```
self.symbol = self.AddIndex("VIX", market=Market.USA).Symbol
```
PY

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.symbol = self.AddIndex("VIX", fillForward=False).Symbol
```
PY

Properties

The `AddIndex` method returns an `Index` object, which have the following properties:

Index

Handling Data

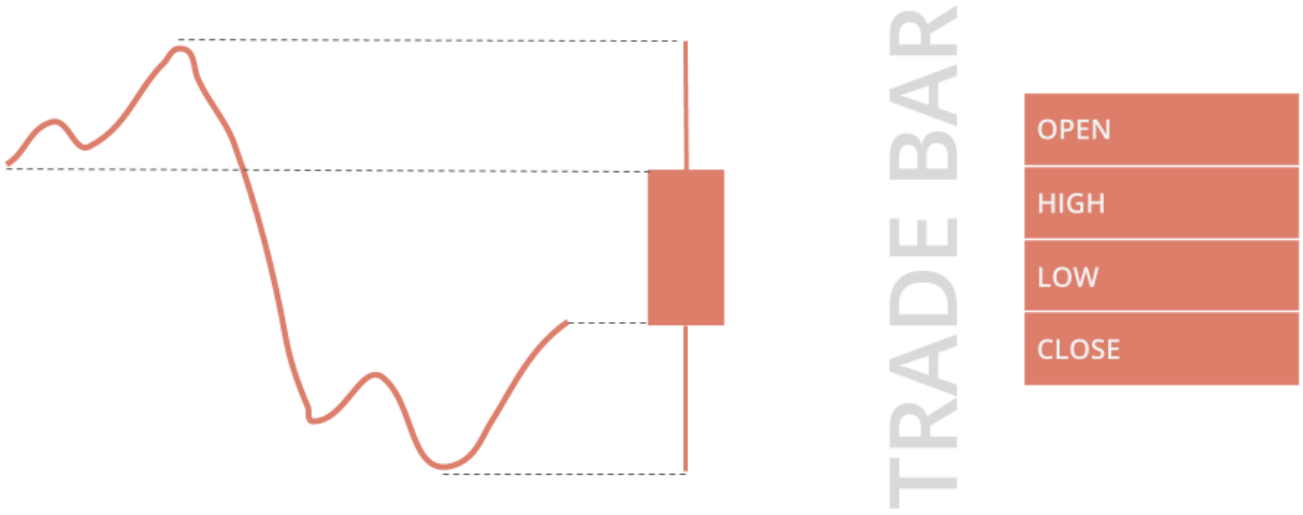
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `Ticks DataDictionary` is made up of `Tick` objects. To access individual data points in the dictionary, you can index the dictionary with the Index ticker or `Symbol` , but we recommend you use the `Symbol` .

Bars

You can't trade Indices, but `TradeBar` objects are bars that represent the open, high, low, and close of an Index price over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the Index `Symbol` . The `Slice` may not contain data for your `Symbol` at every time step. To avoid issues, check if the `Slice` contains data for your Index before you index the `Slice` with the Index `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        value = trade_bar.Value
```

Ticks

`Tick` objects represent a price for the Index at a moment in time. `Tick` objects have the following properties:

Index ticks have a non-zero value for the `Price` property, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . The `Slice` may not contain data for your `Symbol` at every time step. To avoid issues, check if the `Slice` contains data for your Index before you index the `Slice` with the Index `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            value = tick.Value
```

Index

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the US Indices market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Indices market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:20:00
Tuesday	08:30:00 to 15:20:00
Wednesday	08:30:00 to 15:20:00
Thursday	08:30:00 to 15:20:00
Friday	08:30:00 to 15:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	

Early Closes

The following table shows the early closes for the US Indices market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
1999-11-26	13:00:00
2000-07-03	13:00:00
2000-11-24	13:00:00
2001-07-03	13:00:00
2001-11-23	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00
2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-07-03	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00
2018-07-03	13:00:00
2018-11-23	13:00:00
2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The US Indices market trades in the following time zones:

- `America/Chicago`

- America/New York

Assets With Other Hours

The following table shows the indices that have different trading periods than the overall US Indices market:

Symbol	Name
NDX	Nasdaq 100 Index
SPX	S&P 500 Index
VIX	CBOE Volatility Index

Market Hours

NDX

Introduction

This page shows the trading hours, holidays, and time zone of the Nasdaq 100 Index market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nasdaq 100 Index market:

Weekday	Time (America/New York)
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Nasdaq 100 Index market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-06-19	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Nasdaq 100 Index market trades in the `America/New York` time zone.

Market Hours

SPX

Introduction

This page shows the trading hours, holidays, and time zone of the S&P 500 Index market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the S&P 500 Index market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the S&P 500 Index market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-06-19	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The S&P 500 Index market trades in the `America/Chicago` time zone.

Market Hours

VIX

Introduction

This page shows the trading hours, holidays, and time zone of the CBOE Volatility Index market.

Pre-market Hours

The following table shows the pre-market hours for the CBOE Volatility Index market:

Weekday	Time (America/Chicago)
Monday	02:15:00 to 08:15:00
Tuesday	02:15:00 to 08:15:00
Wednesday	02:15:00 to 08:15:00
Thursday	02:15:00 to 08:15:00
Friday	02:15:00 to 08:15:00

Regular Trading Hours

The following table shows the regular trading hours for the CBOE Volatility Index market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

The following table shows the post-market hours for the CBOE Volatility Index market:

Weekday	Time (America/Chicago)
Monday	15:15:00 to 16:00:00
Tuesday	15:15:00 to 16:00:00
Wednesday	15:15:00 to 16:00:00
Thursday	15:15:00 to 16:00:00
Friday	15:15:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the CBOE Volatility Index market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05

Date (*yyyy-mm-dd*)				
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-06-19	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CBOE Volatility Index market trades in the `America/Chicago` time zone.

Asset Classes

Index Options

Index Options are a financial derivative that gives the holder the right (but not the obligation) to buy or sell the value of an underlying Index, such as the S&P 500 index, at the stated exercise price. No actual assets are bought or sold.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateIndexOptionsAlgorithm.py
BasicTemplateIndexOptionsAlgorithm.cs

Index Options

Requesting Data

Introduction

Request Index Options data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Index Options dataset listing . To trade Index Options live, you can use one of the brokerage data feeds . We currently only support European-style Options for Index Options.

Create Subscriptions

Before you can subscribe to an Index Option contract, you must configure the underlying Index and get the contract `Symbol` .

Configure the Underlying Index

In most cases, you should subscribe to the underlying Index before you subscribe to an Index Option contract.

```
self.symbol = self.AddIndex("SPX").Symbol
```

If you subscribe to an Index Option contract but don't have a subscription to the underlying Index, LEAN automatically subscribes to the underlying Index and sets its fill forward property to match that of the Index Option contract. In this case, you still need the Index `Symbol` to subscribe to Index Option contracts. If you don't have access to it, create it.

```
self.symbol = Symbol.Create("SPX", SecurityType.Index, Market.USA)
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Get Contract Symbols

To get Index Option contract `Symbol` objects, call the `CreateOption` method or use the `OptionChainProvider` . If you use the `CreateOption` method, you need to know the specific contract details.

```
# Standard contracts
self.contract_symbol = Symbol.CreateOption(self.symbol, Market.USA,
    OptionStyle.European, OptionRight.Call, 3650, datetime(2022, 6, 17))

# Weekly contracts
self.weekly_contract_symbol = Symbol.CreateOption(self.symbol, "SPXW", Market.USA,
    OptionStyle.European, OptionRight.Call, 3650, datetime(2022, 6, 17))
```

Another way to get an Index Option contract `Symbol` is to use the `OptionChainProvider` . The `GetOptionContractList` method of `OptionChainProvider` returns a list of `Symbol` objects that reference the available Option contracts for a

given underlying Index on a given date. To filter and select contracts, you can use the following properties of each `Symbol` object:

Property	Description
`ID.Date`	The expiration date of the contract.
`ID.StrikePrice`	The strike price of the contract.
`ID.OptionRight`	The contract type. The `OptionRight` enumeration has the following members:
`ID.OptionStyle`	The contract style. The `OptionStyle` enumeration has the following members: We currently only support European-style Options for Index Options.

PY

```
# Standard contracts
self.canonical_symbol = Symbol.CreateCanonicalOption(self.symbol, Market.USA, "?SPX")
contract_symbols = self.OptionChainProvider.GetOptionContractList(self.canonical_symbol, self.Time)
expiry = min([symbol.ID.Date for symbol in contract_symbols])
filtered_symbols = [symbol for symbol in contract_symbols if symbol.ID.Date == expiry and
symbol.ID.OptionRight == OptionRight.Call]
self.contract_symbol = sorted(filtered_symbols, key=lambda symbol: symbol.ID.StrikePrice)[0]

# Weekly contracts
self.weekly_canonical_symbol = Symbol.CreateCanonicalOption(self.symbol, "SPXW", Market.USA, "?SPXW")
weekly_contract_symbols = self.OptionChainProvider.GetOptionContractList(self.weekly_canonical_symbol,
self.Time)
weekly_contract_symbols = [symbol for symbol in weekly_contract_symbols if OptionSymbol.IsWeekly(symbol)]
weekly_expiry = min([symbol.ID.Date for symbol in weekly_contract_symbols])
weekly_filtered_symbols = [symbol for symbol in weekly_contract_symbols if symbol.ID.Date == weekly_expiry
and symbol.ID.OptionRight == OptionRight.Call]
self.weekly_contract_symbol = sorted(weekly_filtered_symbols, key=lambda symbol: symbol.ID.StrikePrice)[0]
```

Subscribe to Contracts

To create an Index Option contract subscription, pass the contract `Symbol` to the `AddIndexOptionContract` method. Save a reference to the contract `Symbol` so you can easily access the contract in the OptionChain that LEAN passes to the `OnData` method. To override the default pricing model of the Option, set a pricing model .

PY

```
option = self.AddIndexOptionContract(self.contract_symbol)
option.PriceModel = OptionPriceModels.BlackScholes()
```

The `AddIndexOptionContract` method creates a subscription for a single Index Option contract and adds it to your **user-defined** universe. To create a dynamic universe of Index Option contracts, add an Index Option universe .

Warm Up Contract Prices

If you subscribe to an Index Option contract with `AddIndexOptionContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
seeder = FuncSecuritySeeder(self.GetLastKnownPrices)
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel, seeder, self))
```

Supported Assets

To view the supported assets in the US Index Options dataset, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Index Option contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Index Option subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddIndexOptionContract` method.

```
self.AddIndexOptionContract(self.contract_symbol, Resolution.Hour)
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Index Options:

You don't need to pass a `Market` argument to the `AddIndexOptionContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.AddIndexOptionContract(self.contract_symbol, fillForward=False)
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Options are

already leveraged products, so you can't change their leverage.

Remove Subscriptions

To remove a contract subscription that you created with `AddIndexOptionContract` , call the `RemoveOptionContract` method. This method is an alias for `RemoveSecurity` .

```
self.RemoveOptionContract(self.contract_symbol)
```

The `RemoveOptionContract` method cancels your open orders for the contract and liquidates your holdings.

Properties

The `AddIndexOptionContract` method returns an `Option` object, which have the following properties:

Helper Methods

The `Option` object provides methods you can use for basic calculations. These methods require the underlying price. To get the `Option` object and the `Security` object for its underlying in any function, use the Option `Symbol` to access the value in the `Securities` object.

```
option = self.Securities[self.contract_symbol]
underlying = self.Securities[self.contract_symbol.Underlying]
underlying_price = underlying.Price
```

To get the Option payoff , call the `GetPayOff` method.

```
pay_off = option.GetPayOff(underlying_price)
```

To get the Option intrinsic value , call the `GetIntrinsicValue` method.

```
intrinsic_value = option.GetIntrinsicValue(underlying_price)
```

To get the Option out-of-the-money amount , call the `OutOfTheMoneyAmount` method.

```
otm_amount = option.OutOfTheMoneyAmount(underlying_price)
```

To check whether the Option can be automatic exercised, call the `IsAutoExercised` method.

```
is_auto_exercised = option.IsAutoExercised(underlying_price)
```

Index Options

Handling Data

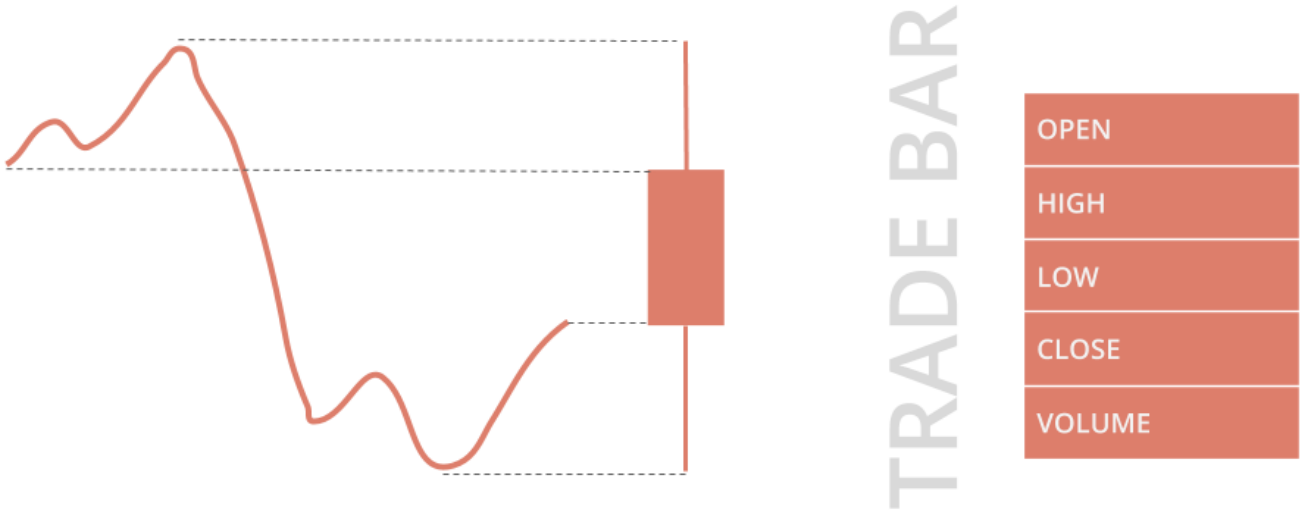
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the contract ticker or `Symbol` , but we recommend you use the `Symbol` .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



`TradeBar` objects have the following properties:

To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

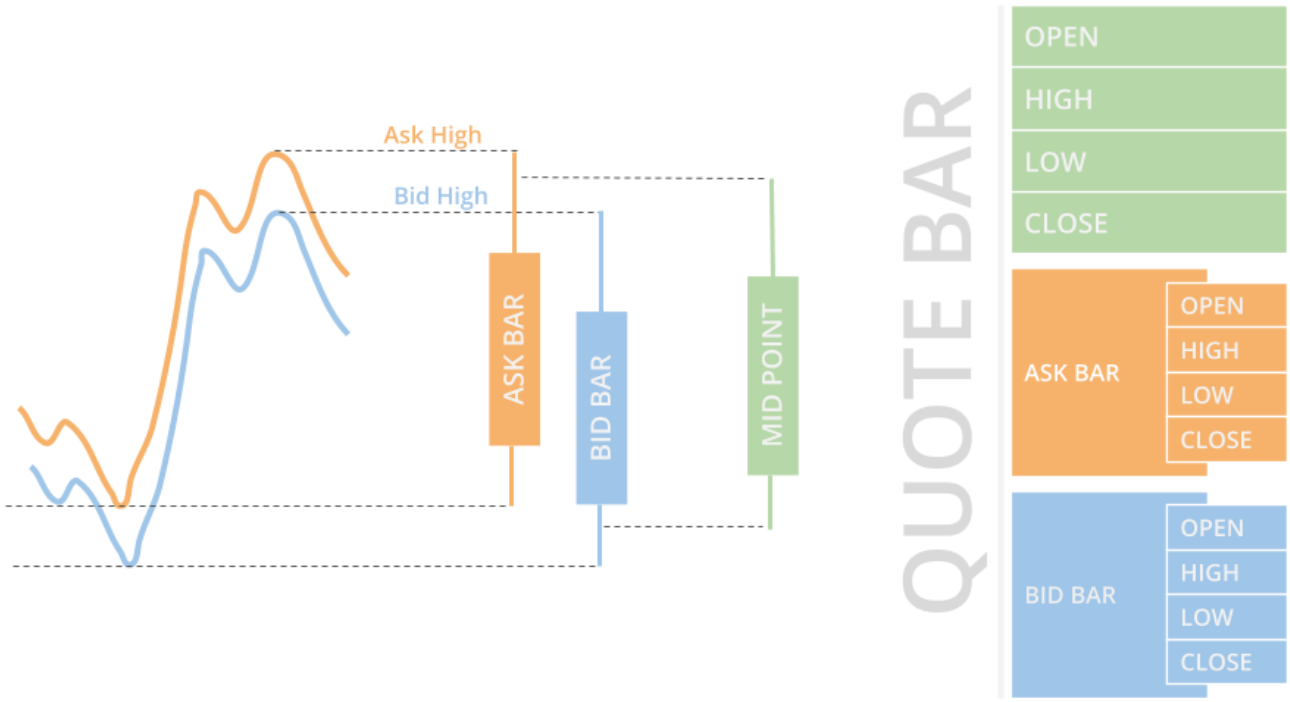
```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.Bars:
        trade_bar = slice.Bars[self.contract_symbol]
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        close_price = trade_bar.Close
```

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively get quotes or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.contract_symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.contract_symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security. They have the following properties:

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contracts = chain.Contracts
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.OptionChains.items():
        contracts = chain.Contracts
```

Option Contracts

`OptionContract` objects represent the data of a single Option contract in the market. They have the following properties:

To get the Option contracts in the `Slice` , use the `Contracts` property of the `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            price = contract.Price
```

Greeks and Implied Volatility

To get the Greeks and implied volatility of an Option contract, use the `Greeks` and `ImpliedVolatility` members.

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            delta = contract.Greeks.Delta
            iv = contract.ImpliedVolatility
```

LEAN only calculates Greeks and implied volatility when you request them because they are expensive operations. If you invoke the `Greeks` member, the Greeks aren't calculated. However, if you invoke the `Greeks.Delta` member, LEAN calculates the delta. To avoid unecessary computation in your algorithm, only request the Greeks and implied volatility when you need them. For more information about the Greeks and implied volatility, see Options Pricing .

Open Interest

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Option` or `OptionContract` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.contract_symbol.Canonical)
    if chain:
        contract = chain.Contracts.get(self.contract_symbol)
        if contract:
            open_interest = contract.OpenInterest
```

Index Options

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the US Index Option market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Index Option market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:20:00
Tuesday	08:30:00 to 15:20:00
Wednesday	08:30:00 to 15:20:00
Thursday	08:30:00 to 15:20:00
Friday	08:30:00 to 15:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Index Option market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	

Early Closes

The following table shows the early closes for the US Index Option market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
1999-11-26	13:00:00
2000-07-03	13:00:00
2000-11-24	13:00:00
2001-07-03	13:00:00
2001-11-23	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00
2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-07-03	13:00:00

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00
2018-07-03	13:00:00
2018-11-23	13:00:00
2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The US Index Option market trades in the following time zones:

- `America/Chicago`

- America/New York

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall US Index Option market:

Symbol	Name
NDX	Nasdaq 100 Index
SPX	S&P 500 Index
VIX	CBOE Volatility Index

Market Hours

NDX

Introduction

This page shows the trading hours, holidays, and time zone of the Nasdaq 100 Index Option contracts market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nasdaq 100 Index Option contracts market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:15:00
Tuesday	09:30:00 to 16:15:00
Wednesday	09:30:00 to 16:15:00
Thursday	09:30:00 to 16:15:00
Friday	09:30:00 to 16:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Nasdaq 100 Index Option contracts market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

Date (*yyyy-mm-dd*)				
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

Date (*yyyy-mm-dd*)				
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-06-19	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Nasdaq 100 Index Option contracts market trades in the `America/New York` time zone.

Market Hours

SPX

Introduction

This page shows the trading hours, holidays, and time zone of the S&P 500 Index Option contracts market.

Pre-market Hours

The following table shows the pre-market hours for the S&P 500 Index Option contracts market:

Weekday	Time (America/Chicago)
Sunday	19:15:00 to 1:00:00:00
Monday	00:00:00 to 08:15:00
Tuesday	00:00:00 to 08:15:00
Wednesday	00:00:00 to 08:15:00
Thursday	00:00:00 to 08:15:00
Friday	00:00:00 to 08:15:00

Regular Trading Hours

The following table shows the regular trading hours for the S&P 500 Index Option contracts market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

The following table shows the post-market hours for the S&P 500 Index Option contracts market:

Weekday	Time (America/Chicago)
Monday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Tuesday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Wednesday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Thursday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Friday	15:15:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the S&P 500 Index Option contracts market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05

Date (*yyyy-mm-dd*)				
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-06-19	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The S&P 500 Index Option contracts market trades in the `America/Chicago` time zone.

Market Hours

VIX

Introduction

This page shows the trading hours, holidays, and time zone of the CBOE Volatility Index Option contracts market.

Pre-market Hours

The following table shows the pre-market hours for the CBOE Volatility Index Option contracts market:

Weekday	Time (America/Chicago)
Sunday	19:15:00 to 1:00:00:00
Monday	00:00:00 to 08:15:00
Tuesday	00:00:00 to 08:15:00
Wednesday	00:00:00 to 08:15:00
Thursday	00:00:00 to 08:15:00
Friday	00:00:00 to 08:15:00

Regular Trading Hours

The following table shows the regular trading hours for the CBOE Volatility Index Option contracts market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

The following table shows the post-market hours for the CBOE Volatility Index Option contracts market:

Weekday	Time (America/Chicago)
Monday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Tuesday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Wednesday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Thursday	15:15:00 to 16:00:00, 19:15:00 to 1:00:00:00
Friday	15:15:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the CBOE Volatility Index Option contracts market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

Date (*yyyy-mm-dd*)				
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05

Date (*yyyy-mm-dd*)				
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-06-19	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CBOE Volatility Index Option contracts market trades in the `America/Chicago` time zone.

Asset Classes

CFD

A contract for difference (CFD) is a contract between a buyer and a seller that stipulates that the buyer must pay the seller the difference between the current value of an asset and its value at contract time.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateCfdAlgorithm.py
BasicTemplateCfdAlgorithm.cs

CFD

Requesting Data

Introduction

Request Contract for Difference (CFD) data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the CFD dataset listing . To trade CFDs live, you can use our CFD data feed .

Create Subscriptions

To create a CFD subscription, in the `Initialize` method, call the `AddCfd` method. The `AddCfd` method returns a `Cfd` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
PY
self.symbol = self.AddCfd("XAUUSD").Symbol
```

To view the supported CFD contracts, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for CFD subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				✓
`Second`		✓		
`Minute`		✓		
`Hour`		✓		
`Daily`		✓		

The default resolution for CFD subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddCfd` method.

```
PY
self.symbol = self.AddCfd("XAUUSD", Resolution.Daily).Symbol
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The only market available for CFD contracts is `Market.Oanda` , so you don't need to pass a `market` argument to the `AddCfd` method.

```
self.symbol = self.AddCfd("XAUUSD", market=Market.Oanda).Symbol
```
PY

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
self.symbol = self.AddCfd("XAUUSD", fillForward=False).Symbol
```
PY

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The Oanda brokerage let's you use up to 50x leverage on margin accounts. To change the amount of leverage you can use for a CFD contract, pass a `leverage` argument to the `AddCfd` method.

```
self.symbol = self.AddCfd("XAUUSD", leverage=35).Symbol
```
PY

Properties

The `AddCfd` method returns a `Cfd` object, which have the following properties:

CFD

Handling Data

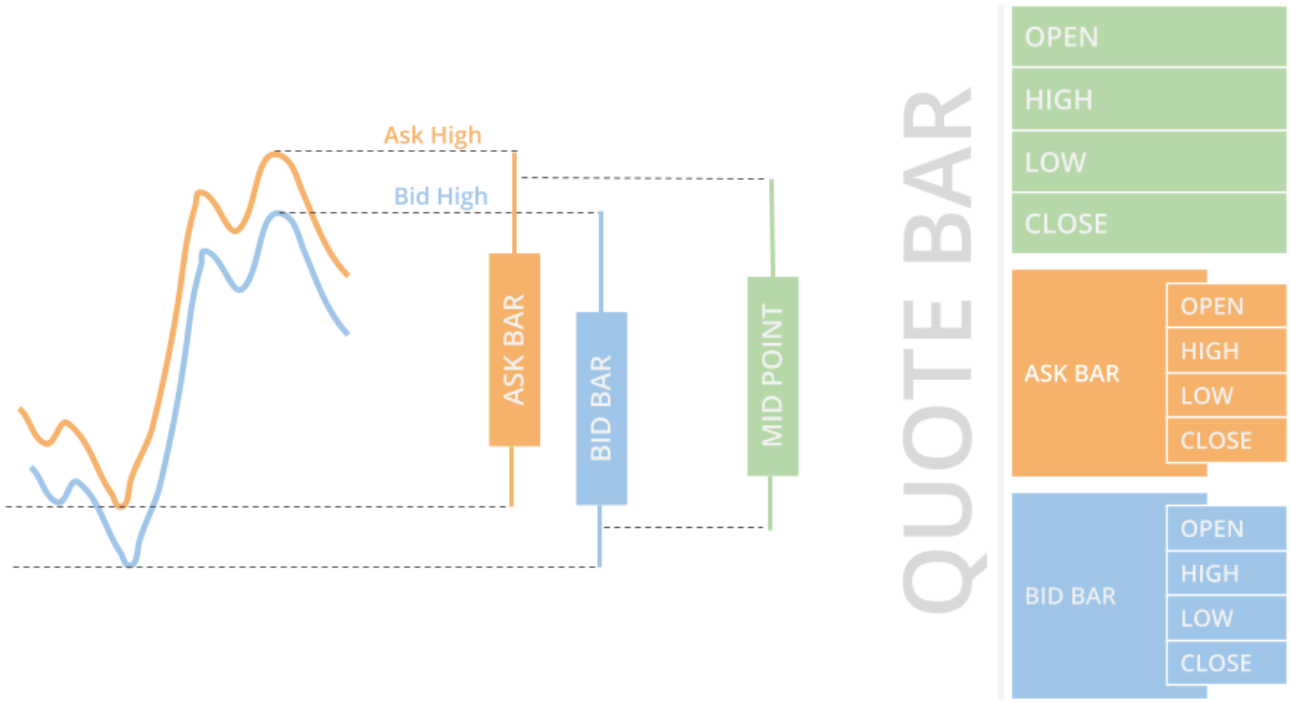
Introduction

LEAN passes the data you request to the `OnData` method so you can make trading decisions. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `QuoteBars DataDictionary` is made up of `QuoteBar` objects. To access individual data points in the dictionary, you can index the dictionary with the CFD ticker or `Symbol` , but we recommend you use the `Symbol` .

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



`QuoteBar` objects have the following properties:

To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the CFD `Symbol` . If the CFD doesn't actively get quotes or you are in the same time step as when you added the CFD subscription, the `Slice` may

not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your CFD before you index the `Slice` with the CFD `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        ask_price = quote_bar.Ask.Close
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the CFD. A quote tick is an offer to buy or sell the CFD at a specific price. `Tick` objects have the following properties:

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the CFD doesn't actively trade or you are in the same time step as when you added the CFD subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your CFD before you index the `Slice` with the CFD `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            price = tick.Price
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `list[Tick]` objects.

```
def OnData(self, slice: Slice) -> None:
    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            price = tick.Price
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

CFD

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CFD market:

Weekday	Time (America/New York)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)			
2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

The following table shows the early closes for the CFD market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2018-12-31	17:00:00

Late Opens

There are no days with late opens.

Time Zone

The CFD market trades in the following time zones:

- `America/Chicago`
- `America/New York`
- `Asia/Hong Kong`
- `Asia/Singapore`
- `Australia/Sydney`
- `Europe/Amsterdam`
- `Europe/Berlin`
- `Europe/London`
- `Europe/Paris`
- `Europe/Zurich`
- `UTC`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CFD market:

Symbol	Name
AU200AUD	Australia 200
BCOUSD	Brent Crude Oil
CH20CHF	Swiss 20
CORNUSD	Corn
DE10YBEUR	Bund
DE30EUR	Germany 30
EU50EUR	Europe 50
FR40EUR	France 40
HK33HKD	Hong Kong 33
JP225USD	Japan 225
NAS100USD	US Nas 100
NATGASUSD	Natural Gas
NL25EUR	Netherlands 25

Symbol	Name
SG30SGD	Singapore 30
SOYBNUSD	Soybeans
SPX500USD	US SPX 500
SUGARUSD	Sugar
UK100GBP	UK 100
UK10YBGBP	UK 10Y Gilt
US2000USD	US Russ 2000
US30USD	US Wall St 30
USB02YUSD	US 2Y T-Note
USB05YUSD	US 5Y T-Note
USB10YUSD	US 10Y T-Note
USB30YUSD	US T-Bond
WHEATUSD	Wheat
WTICOUSD	West Texas Oil
XAGAUD	Silver/AUD
XAGCAD	Silver/CAD
XAGCHF	Silver/CHF
XAGEUR	Silver/EUR
XAGGBP	Silver/GBP
XAGHKD	Silver/HKD
XAGJPY	Silver/JPY
XAGNZD	Silver/NZD
XAGSGD	Silver/SGD
XAGUSD	Silver
XAUAUD	Gold/AUD
XAUCAD	Gold/CAD

Symbol	Name
XAUCHF	Gold/CHF
XAUEUR	Gold/EUR
XAUGBP	Gold/GBP
XAUHKD	Gold/HKD
XAUJPY	Gold/JPY
XAUNZD	Gold/NZD
XAUSGD	Gold/SGD
XAUUSD	Gold
XAUXAG	Gold/Silver
XCUUSD	Copper
XPDUSD	Palladium
XPTUSD	Platinum

Market Hours

AU200AUD

Introduction

This page shows the trading hours, holidays, and time zone of the Australia 200 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Australia 200 contract in the CFD market:

Weekday	Time (Australia/Sydney)
Monday	09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Tuesday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Wednesday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Thursday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Friday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Australia 200 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2004-04-09	2004-12-26	2004-12-27	2005-03-25	2005-12-25
2005-12-26	2006-04-14	2006-12-25	2007-01-26	2007-04-06
2007-12-25	2008-03-21	2008-04-25	2008-12-25	2008-12-26
2009-04-10	2009-12-25	2010-01-01	2010-04-02	2010-12-26
2010-12-27	2011-04-22	2011-04-25	2011-12-25	2011-12-26
2011-12-27	2012-01-02	2012-01-26	2012-04-06	2012-04-09
2012-04-25	2012-12-24	2012-12-25	2012-12-31	2013-01-28
2013-03-29	2013-12-25	2014-01-27	2014-04-18	2014-04-25
2014-12-25	2014-12-26	2015-04-03	2015-12-24	2015-12-25
2015-12-31	2016-01-01	2016-03-25	2016-12-23	2016-12-25
2016-12-26	2016-12-30	2017-04-14	2017-12-25	2018-03-30
2018-12-24	2018-12-25	2018-12-31	2019-01-01	2019-01-28
2019-04-19	2019-04-22	2019-04-25	2019-06-10	2019-12-25
2019-12-26	2020-01-01	2020-01-27	2020-04-10	2020-04-13
2020-04-25	2020-06-08	2020-12-25	2020-12-26	2020-12-28
2021-01-01	2021-01-26	2021-04-02	2021-04-05	2021-04-26
2021-06-14	2021-12-25	2021-12-26	2021-12-27	2021-12-28
2022-01-03	2022-01-26	2022-04-15	2022-04-18	2022-04-25
2022-06-13	2022-12-25	2022-12-26	2022-12-27	2023-01-02
2023-01-26	2023-04-07	2023-04-10	2023-04-25	2023-06-12
2023-12-25	2023-12-26	2024-01-01	2024-01-26	2024-03-29
2024-04-01	2024-04-25	2024-06-10		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Australia 200 contract in the CFD market trades in the `Australia/Sydney` time zone.

Market Hours

BCOUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Crude Oil contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Brent Crude Oil contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	20:00:00 to 24:00:00
Monday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Brent Crude Oil contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-12-25	2018-01-01	2018-03-30	2018-12-25
2019-01-01	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-04-02	2022-04-15	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Brent Crude Oil contract in the CFD market trades in the `America/New York` time zone.

Market Hours

CH20CHF

Introduction

This page shows the trading hours, holidays, and time zone of the Swiss 20 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Swiss 20 contract in the CFD market:

Weekday	Time (Europe/Zurich)
Monday	08:00:00 to 22:00:00
Tuesday	08:00:00 to 22:00:00
Wednesday	08:00:00 to 22:00:00
Thursday	08:00:00 to 22:00:00
Friday	08:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Swiss 20 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-04-14	2017-04-17	2017-05-01	2017-05-25
2017-06-05	2017-08-01	2017-12-25	2017-12-26	2018-01-01
2018-03-30	2018-04-02	2018-05-01	2018-05-10	2018-05-21
2018-08-01	2018-12-25	2018-12-26	2019-01-01	2019-04-19
2019-04-22	2019-05-01	2019-05-30	2019-06-10	2019-08-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-06-01	2020-12-25	2021-01-01	2021-04-02
2021-04-05	2021-05-13	2021-05-24	2021-12-24	2021-12-31
2022-04-15	2022-04-18	2022-05-26	2022-06-06	2022-08-01
2022-12-26	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Swiss 20 contract in the CFD market trades in the `Europe/Zurich` time zone.

Market Hours

CORNUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Corn contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Corn contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Tuesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Wednesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Thursday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Friday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Corn contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Corn contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

DE10YBEUR

Introduction

This page shows the trading hours, holidays, and time zone of the Bund contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Bund contract in the CFD market:

Weekday	Time (Europe/Berlin)
Monday	01:15:00 to 22:00:00
Tuesday	01:15:00 to 22:00:00
Wednesday	01:15:00 to 22:00:00
Thursday	01:15:00 to 22:00:00
Friday	01:15:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Bund contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-04-17	2017-05-01	2017-06-05	2017-10-03
2017-10-31	2017-12-25	2017-12-26	2018-01-01	2018-03-30
2018-04-02	2018-05-01	2018-05-21	2018-10-03	2018-10-31
2018-12-25	2018-12-26	2019-01-01	2019-04-19	2019-04-22
2019-05-01	2019-06-10	2019-10-03	2019-10-31	2019-12-25
2019-12-26	2020-01-01	2020-04-10	2020-04-13	2020-05-01
2020-06-01	2020-12-25	2021-01-01	2021-04-02	2021-04-05
2021-05-01	2021-05-24	2021-12-24	2021-12-31	2022-04-15
2022-04-18	2022-05-01	2022-06-06	2022-10-03	2022-10-31
2022-12-26	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Bund contract in the CFD market trades in the `Europe/Berlin` time zone.

Market Hours

DE30EUR

Introduction

This page shows the trading hours, holidays, and time zone of the Germany 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Germany 30 contract in the CFD market:

Weekday	Time (Europe/Berlin)
Monday	01:15:00 to 22:00:00
Tuesday	01:15:00 to 22:00:00
Wednesday	01:15:00 to 22:00:00
Thursday	01:15:00 to 22:00:00
Friday	01:15:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Germany 30 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-04-17	2017-05-01	2017-06-05	2017-10-03
2017-10-31	2017-12-25	2017-12-26	2018-01-01	2018-03-30
2018-04-02	2018-05-01	2018-05-21	2018-10-03	2018-10-31
2018-12-25	2018-12-26	2019-01-01	2019-04-19	2019-04-22
2019-05-01	2019-06-10	2019-10-03	2019-10-31	2019-12-25
2019-12-26	2020-01-01	2020-04-10	2020-04-13	2020-05-01
2020-06-01	2020-12-25	2021-01-01	2021-04-02	2021-04-05
2021-05-01	2021-05-24	2021-12-24	2021-12-31	2022-04-15
2022-04-18	2022-05-01	2022-06-06	2022-10-03	2022-10-31
2022-12-26	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Germany 30 contract in the CFD market trades in the `Europe/Berlin` time zone.

Market Hours

EU50EUR

Introduction

This page shows the trading hours, holidays, and time zone of the Europe 50 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Europe 50 contract in the CFD market:

Weekday	**Time (Europe/Berlin)**
Monday	01:15:00 to 22:00:00
Tuesday	01:15:00 to 22:00:00
Wednesday	01:15:00 to 22:00:00
Thursday	01:15:00 to 22:00:00
Friday	01:15:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Europe 50 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-04-21	2003-05-01	2003-12-25	2003-12-26
2004-01-01	2004-04-09	2004-04-12	2004-12-26	2005-03-25
2005-03-28	2005-05-01	2005-12-25	2005-12-26	2006-01-01
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26
2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-25
2007-12-26	2008-01-01	2008-03-21	2008-03-24	2008-05-01

Date (*yyyy-mm-dd*)				
2008-12-25	2008-12-26	2009-01-01	2009-04-10	2009-04-13
2009-05-01	2009-12-25	2010-01-01	2010-04-02	2010-04-05
2010-12-26	2011-04-22	2011-04-25	2011-05-01	2011-12-25
2011-12-26	2012-01-01	2012-04-06	2012-04-09	2012-05-01
2012-12-25	2013-01-01	2013-03-29	2013-04-01	2013-05-01
2013-12-25	2013-12-26	2014-01-01	2014-04-18	2014-04-21
2014-05-01	2014-12-25	2014-12-26	2015-01-01	2015-04-03
2015-04-06	2015-05-01	2015-12-25	2016-01-01	2016-03-25
2016-03-28	2016-05-01	2016-12-25	2016-12-26	2017-01-01
2017-04-14	2017-04-14	2017-04-17	2017-05-01	2017-12-25
2017-12-26	2018-01-01	2018-03-30	2018-04-02	2018-05-01
2018-12-25	2018-12-26	2019-01-01	2019-04-19	2019-04-22
2019-05-01	2019-12-25	2019-12-26	2020-01-01	2020-04-10
2020-04-13	2020-05-01	2020-12-25	2020-12-26	2021-01-01
2021-04-02	2021-04-05	2021-05-01	2021-12-25	2021-12-26
2022-01-01	2022-04-15	2022-04-18	2022-05-01	2022-12-25
2022-12-26	2023-01-01	2023-01-02	2023-04-07	2023-04-10
2023-05-01	2023-12-25	2023-12-26	2024-01-01	2024-03-29
2024-04-01	2024-05-01			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Europe 50 contract in the CFD market trades in the `Europe/Berlin` time zone.

Market Hours

FR40EUR

Introduction

This page shows the trading hours, holidays, and time zone of the France 40 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the France 40 contract in the CFD market:

Weekday	**Time (Europe/Paris)**
Monday	08:00:00 to 22:00:00
Tuesday	08:00:00 to 22:00:00
Wednesday	08:00:00 to 22:00:00
Thursday	08:00:00 to 22:00:00
Friday	08:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the France 40 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-04-21	2003-05-01	2003-12-25	2003-12-26
2004-01-01	2004-04-09	2004-04-12	2004-12-26	2005-03-25
2005-03-28	2005-05-01	2005-12-25	2005-12-26	2006-01-01
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26
2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-25
2007-12-26	2008-01-01	2008-03-21	2008-03-24	2008-05-01

Date (*yyyy-mm-dd*)				
2008-12-25	2008-12-26	2009-01-01	2009-04-10	2009-04-13
2009-05-01	2009-12-25	2010-01-01	2010-04-02	2010-04-05
2010-12-26	2011-04-22	2011-04-25	2011-05-01	2011-12-25
2011-12-26	2012-01-01	2012-04-06	2012-04-09	2012-05-01
2012-12-25	2013-01-01	2013-03-29	2013-04-01	2013-05-01
2013-12-25	2013-12-26	2014-01-01	2014-04-18	2014-04-21
2014-05-01	2014-12-25	2014-12-26	2015-01-01	2015-04-03
2015-04-06	2015-05-01	2015-12-25	2016-01-01	2016-03-25
2016-03-28	2016-05-01	2016-12-25	2016-12-26	2017-01-01
2017-04-14	2017-04-17	2017-05-01	2017-12-25	2017-12-26
2018-01-01	2018-03-30	2018-04-02	2018-05-01	2018-12-25
2018-12-26	2019-01-01	2019-04-19	2019-04-22	2019-05-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-12-25	2020-12-26	2021-01-01	2021-04-02
2021-04-05	2021-05-01	2021-12-25	2021-12-26	2022-01-01
2022-04-15	2022-04-18	2022-05-01	2022-12-25	2022-12-26
2023-01-01	2023-01-02	2023-04-07	2023-04-10	2023-05-01
2023-12-25	2023-12-26	2024-01-01	2024-03-29	2024-04-01
2024-05-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The France 40 contract in the CFD market trades in the `Europe/Paris` time zone.

Market Hours

HK33HKD

Introduction

This page shows the trading hours, holidays, and time zone of the Hong Kong 33 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Hong Kong 33 contract in the CFD market:

Weekday	**Time (Asia/Hong Kong)**
Monday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Hong Kong 33 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-01-31	2003-02-02	2003-02-03	2003-04-18	2003-04-21
2003-05-01	2003-05-08	2003-06-04	2003-07-01	2003-09-12
2003-10-01	2003-12-25	2003-12-26	2004-01-01	2004-01-22
2004-01-23	2004-04-05	2004-04-09	2004-04-12	2004-05-26
2004-06-22	2004-07-01	2004-09-29	2004-10-01	2004-10-22
2004-12-26	2004-12-27	2005-02-09	2005-02-10	2005-02-11

Date (*yyyy-mm-dd*)				
2005-03-25	2005-03-28	2005-04-05	2005-05-02	2005-05-16
2005-07-01	2005-09-19	2005-10-11	2005-12-25	2005-12-26
2005-12-27	2006-01-02	2006-01-29	2006-01-30	2006-01-31
2006-04-05	2006-04-14	2006-04-17	2006-05-01	2006-05-05
2006-05-31	2006-10-02	2006-10-30	2006-12-25	2006-12-26
2007-01-01	2007-02-18	2007-02-19	2007-02-20	2007-04-05
2007-04-06	2007-04-09	2007-05-01	2007-05-24	2007-06-19
2007-07-02	2007-09-26	2007-10-01	2007-10-19	2007-12-25
2007-12-26	2008-01-01	2008-02-07	2008-02-08	2008-03-21
2008-03-24	2008-04-04	2008-05-01	2008-05-12	2008-06-09
2008-07-01	2008-08-06	2008-08-22	2008-09-15	2008-10-01
2008-10-07	2008-12-25	2008-12-26	2009-01-01	2009-01-26
2009-01-27	2009-01-28	2009-04-10	2009-04-13	2009-05-01
2009-05-28	2009-07-01	2009-10-01	2009-10-04	2009-10-26
2009-12-25	2010-01-01	2010-02-14	2010-02-15	2010-02-16
2010-04-02	2010-04-05	2010-04-06	2010-05-21	2010-06-16
2010-07-01	2010-09-23	2010-10-01	2010-12-26	2010-12-27
2011-02-03	2011-02-04	2011-04-05	2011-04-22	2011-04-25
2011-05-02	2011-05-10	2011-06-06	2011-07-01	2011-09-13
2011-09-29	2011-10-05	2011-12-25	2011-12-26	2011-12-27
2012-01-02	2012-01-23	2012-01-24	2012-01-25	2012-12-25
2013-01-01	2013-02-10	2013-02-11	2013-02-12	2013-02-13
2013-03-29	2013-04-01	2013-04-04	2013-05-01	2013-05-17
2013-06-12	2013-07-01	2013-08-14	2013-09-20	2013-10-01
2013-10-14	2013-12-25	2013-12-26	2014-01-01	2014-01-31
2014-02-02	2014-02-03	2014-04-18	2014-04-21	2014-05-01

Date (*yyyy-mm-dd*)				
2014-05-06	2014-06-02	2014-07-01	2014-09-09	2014-10-01
2014-10-02	2014-12-25	2014-12-26	2015-01-01	2015-02-19
2015-02-20	2015-04-03	2015-04-06	2015-05-01	2015-05-25
2015-12-24	2015-12-25	2015-12-31	2016-01-01	2016-02-08
2016-02-09	2016-03-25	2016-07-01	2016-09-16	2016-12-25
2016-12-26	2017-01-29	2017-01-30	2017-04-14	2017-12-25
2018-02-16	2018-02-18	2018-03-30	2018-12-24	2018-12-25
2018-12-31	2019-01-01	2019-02-05	2019-02-05	2019-02-06
2019-02-06	2019-02-07	2019-04-05	2019-04-19	2019-04-22
2019-05-01	2019-05-13	2019-06-07	2019-07-01	2019-09-14
2019-10-01	2019-10-07	2019-12-25	2019-12-26	2020-01-01
2020-01-25	2020-01-26	2020-01-27	2020-01-28	2020-04-04
2020-04-10	2020-04-13	2020-04-30	2020-05-01	2020-06-25
2020-07-01	2020-10-01	2020-10-02	2020-10-26	2020-12-25
2020-12-26	2021-01-01	2021-02-12	2021-02-13	2021-02-14
2021-02-15	2021-04-02	2021-04-05	2021-04-06	2021-05-01
2021-05-19	2021-06-14	2021-07-01	2021-09-22	2021-10-01
2021-10-14	2021-12-25	2021-12-26	2021-12-27	2022-01-01
2022-02-01	2022-02-02	2022-02-03	2022-04-05	2022-04-15
2022-04-18	2022-05-02	2022-05-09	2022-06-03	2022-07-01
2022-09-11	2022-10-01	2022-10-04	2022-12-25	2022-12-26
2022-12-27	2023-01-02	2023-01-22	2023-01-23	2023-01-24
2023-01-25	2023-04-05	2023-04-07	2023-04-10	2023-05-01
2023-05-26	2023-06-22	2023-07-01	2023-09-30	2023-10-02
2023-10-23	2023-12-25	2023-12-26	2024-01-01	2024-02-10
2024-02-11	2024-02-12	2024-02-13	2024-03-29	2024-04-01

Date (*yyyy-mm-dd*)			
2024-04-04	2024-05-01	2024-05-15	2024-06-10

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Hong Kong 33 contract in the CFD market trades in the `Asia/Hong Kong` time zone.

Market Hours

JP225USD

Introduction

This page shows the trading hours, holidays, and time zone of the Japan 225 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Japan 225 contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Japan 225 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-05-26	2003-07-04	2003-09-01	2003-11-27
2003-12-25	2004-01-01	2004-01-19	2004-02-16	2004-04-09
2004-05-31	2004-06-11	2004-09-06	2004-11-25	2004-12-24
2005-01-17	2005-02-21	2005-03-25	2005-05-30	2005-07-04
2005-12-26	2006-01-02	2006-04-14	2006-12-25	2007-01-01

Date (*yyyy-mm-dd*)				
2007-12-25	2008-01-01	2008-03-21	2008-12-25	2009-01-01
2009-04-10	2009-12-25	2010-01-01	2010-12-24	2011-04-22
2011-12-26	2012-01-02	2012-12-25	2013-01-01	2013-03-29
2013-12-25	2014-01-01	2014-04-18	2015-12-25	2016-01-01
2016-03-25	2017-04-14	2018-03-30	2019-01-01	2019-01-02
2019-01-03	2019-01-14	2019-02-11	2019-03-21	2019-04-29
2019-05-03	2019-05-04	2019-05-06	2019-07-15	2019-08-12
2019-09-16	2019-09-23	2019-10-14	2019-11-04	2019-11-23
2019-12-23	2019-12-31	2020-01-01	2020-01-02	2020-01-03
2020-01-13	2020-02-11	2020-03-20	2020-04-29	2020-05-04
2020-05-04	2020-05-05	2020-07-20	2020-08-11	2020-09-21
2020-09-22	2020-10-12	2020-11-03	2020-11-23	2020-12-23
2020-12-31	2021-01-01	2021-01-02	2021-01-04	2021-01-11
2021-02-11	2021-03-20	2021-04-29	2021-05-03	2021-05-04
2021-05-05	2021-07-19	2021-08-11	2021-09-20	2021-09-23
2021-10-11	2021-11-03	2021-11-23	2021-12-23	2021-12-31
2022-01-01	2022-01-03	2022-01-03	2022-01-10	2022-02-11
2022-03-21	2022-04-29	2022-05-03	2022-05-04	2022-05-05
2022-07-18	2022-08-11	2022-09-19	2022-09-23	2022-10-10
2022-11-03	2022-11-23	2022-12-23	2022-12-25	2022-12-26
2022-12-31	2023-01-01	2023-01-02	2023-01-02	2023-01-03
2023-01-09	2023-02-11	2023-03-21	2023-04-29	2023-05-03
2023-05-04	2023-05-05	2023-07-17	2023-08-11	2023-09-18
2023-09-23	2023-10-09	2023-11-03	2023-11-23	2023-12-23
2024-01-01	2024-01-01	2024-01-02	2024-01-03	2024-01-08
2024-02-12	2024-03-20	2024-04-29	2024-05-03	2024-05-04

Date (*yyyy-mm-dd*)
2024-05-06

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Japan 225 contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

NAS100USD

Introduction

This page shows the trading hours, holidays, and time zone of the US Nas 100 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Nas 100 contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Nas 100 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-07-04	2003-12-25	2004-01-01	2004-04-09
2004-06-11	2004-12-24	2005-03-25	2006-04-14	2006-12-25
2007-01-01	2007-12-25	2008-01-01	2008-03-21	2008-12-25
2009-01-01	2009-04-10	2009-12-25	2010-01-01	2010-12-24
2011-04-22	2011-12-26	2012-01-02	2012-12-25	2013-01-01
2013-03-29	2013-12-25	2014-01-01	2014-04-18	2015-12-25
2016-03-25	2017-04-14	2018-03-30	2019-01-01	2019-04-19
2019-12-25	2020-01-01	2020-04-10	2020-12-25	2021-01-01
2021-04-02	2021-12-24	2022-01-01	2022-04-15	2022-12-25
2022-12-26	2023-01-01	2023-01-02	2023-04-07	2023-12-25
2024-01-01	2024-03-29			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Nas 100 contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

NATGASUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Natural Gas contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Natural Gas contract in the CFD market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Natural Gas contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-29	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-09-06	2021-11-25	2021-12-24	2022-01-17	2022-02-21
2022-04-15	2022-05-30	2022-07-04	2022-09-05	2022-11-24
2022-12-25	2022-12-26	2023-01-01	2023-01-02	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Natural Gas contract in the CFD market trades in the `America/New York` time zone.

Market Hours

NL25EUR

Introduction

This page shows the trading hours, holidays, and time zone of the Netherlands 25 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Netherlands 25 contract in the CFD market:

Weekday	Time (Europe/Amsterdam)
Monday	08:00:00 to 22:00:00
Tuesday	08:00:00 to 22:00:00
Wednesday	08:00:00 to 22:00:00
Thursday	08:00:00 to 22:00:00
Friday	08:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Netherlands 25 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2008-03-21	2008-03-24	2008-05-01	2008-12-25	2008-12-26
2009-01-01	2009-04-10	2009-04-13	2009-05-01	2009-12-25
2010-01-01	2010-04-02	2010-04-05	2010-12-26	2011-04-22
2011-04-25	2011-05-01	2011-12-25	2011-12-26	2012-01-01
2012-04-06	2012-04-09	2012-05-01	2012-12-25	2013-01-01
2013-03-29	2013-04-01	2013-05-01	2013-12-25	2013-12-26
2014-01-01	2014-04-18	2014-04-21	2014-05-01	2014-12-25
2014-12-26	2015-01-01	2015-04-03	2015-04-06	2015-05-01
2015-12-25	2016-01-01	2016-03-25	2016-03-28	2016-05-01
2016-12-25	2016-12-26	2017-01-01	2017-04-14	2017-04-17
2017-05-01	2017-12-25	2017-12-26	2018-01-01	2018-03-30
2018-04-02	2018-05-01	2018-12-25	2018-12-26	2019-01-01
2019-04-19	2019-04-22	2019-05-01	2019-12-25	2019-12-26
2020-01-01	2020-04-10	2020-04-13	2020-05-01	2020-12-25
2020-12-26	2021-01-01	2021-04-02	2021-04-05	2021-05-01
2021-12-25	2021-12-26	2022-01-01	2022-04-15	2022-04-18
2022-05-01	2022-12-25	2022-12-26	2023-01-01	2023-01-02
2023-04-07	2023-04-10	2023-05-01	2023-12-25	2023-12-26
2024-01-01	2024-03-29	2024-04-01	2024-05-01	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Netherlands 25 contract in the CFD market trades in the `Europe/Amsterdam` time zone.

Market Hours

SG30SGD

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Singapore 30 contract in the CFD market:

Weekday	Time (Asia/Singapore)
Monday	08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Tuesday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Wednesday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Thursday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Friday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Singapore 30 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2010-02-15	2011-02-13	2012-01-23	2013-02-11	2014-01-31
2015-02-19	2016-02-08	2017-01-02	2017-01-30	2017-04-14
2017-05-01	2017-05-10	2017-06-26	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-16
2018-03-30	2018-05-01	2018-08-09	2018-12-25	2019-01-01
2019-02-05	2019-02-05	2019-04-19	2019-05-01	2019-08-09
2019-12-25	2020-01-01	2020-01-25	2020-01-27	2020-04-10
2020-05-01	2020-08-10	2020-12-25	2021-01-01	2021-02-12
2021-04-02	2021-05-01	2021-08-09	2021-12-25	2022-01-01
2022-02-01	2022-04-15	2022-05-02	2022-08-09	2022-12-26
2023-01-02				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Singapore 30 contract in the CFD market trades in the `Asia/Singapore` time zone.

Market Hours

SOYBNUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Soybeans contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Soybeans contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Tuesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Wednesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Thursday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Friday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Soybeans contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Soybeans contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

SPX500USD

Introduction

This page shows the trading hours, holidays, and time zone of the US SPX 500 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US SPX 500 contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US SPX 500 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-07-04	2003-12-25	2004-01-01	2004-04-09
2004-06-11	2004-12-24	2005-03-25	2006-04-14	2006-12-25
2007-01-01	2007-12-25	2008-01-01	2008-03-21	2008-12-25
2009-01-01	2009-04-10	2009-12-25	2010-01-01	2010-12-24
2011-04-22	2011-12-26	2012-01-02	2012-12-25	2013-01-01
2013-03-29	2013-12-25	2014-01-01	2014-04-18	2015-12-25
2016-03-25	2017-04-14	2018-03-30	2019-01-01	2019-04-19
2019-12-25	2020-01-01	2020-04-10	2020-12-25	2021-01-01
2021-04-02	2021-12-24	2022-01-01	2022-04-15	2022-12-25
2022-12-26	2023-01-01	2023-01-02	2023-04-07	2023-12-25
2024-01-01	2024-03-29			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US SPX 500 contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

SUGARUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Sugar contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Sugar contract in the CFD market:

Weekday	Time (America/New York)
Monday	03:30:00 to 13:00:00
Tuesday	03:30:00 to 13:00:00
Wednesday	03:30:00 to 13:00:00
Thursday	03:30:00 to 13:00:00
Friday	03:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Sugar contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Sugar contract in the CFD market trades in the `America/New York` time zone.

Market Hours

UK100GBP

Introduction

This page shows the trading hours, holidays, and time zone of the UK 100 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UK 100 contract in the CFD market:

Weekday	**Time (Europe/London)**
Monday	01:00:00 to 21:00:00
Tuesday	01:00:00 to 21:00:00
Wednesday	01:00:00 to 21:00:00
Thursday	01:00:00 to 21:00:00
Friday	01:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the UK 100 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-04-21	2003-05-05	2003-05-26	2003-08-25
2003-12-25	2003-12-26	2004-01-01	2004-04-09	2004-04-12
2004-05-03	2004-05-31	2004-08-30	2004-12-26	2004-12-27
2004-12-28	2005-01-03	2005-03-25	2005-03-28	2005-05-02
2005-05-30	2005-08-29	2005-12-25	2005-12-26	2005-12-27
2006-01-02	2006-04-14	2006-04-17	2006-05-01	2006-05-29

Date (*yyyy-mm-dd*)				
2006-08-28	2006-12-25	2006-12-26	2007-01-01	2007-04-06
2007-04-09	2007-05-07	2007-05-28	2007-08-27	2007-12-25
2007-12-26	2008-01-01	2008-03-21	2008-03-24	2008-05-05
2008-05-26	2008-08-25	2008-12-25	2008-12-26	2009-01-01
2009-04-10	2009-04-13	2009-05-04	2009-05-25	2009-08-31
2009-12-25	2009-12-28	2010-01-01	2010-04-02	2010-04-05
2010-05-03	2010-05-31	2010-08-30	2010-12-26	2010-12-27
2010-12-28	2011-01-03	2011-04-22	2011-04-25	2011-05-02
2011-05-30	2011-08-29	2011-12-25	2011-12-26	2011-12-27
2012-01-02	2012-12-25	2013-01-01	2013-03-29	2013-04-01
2013-05-06	2013-05-27	2013-08-26	2013-12-25	2013-12-26
2014-01-01	2014-04-18	2014-04-21	2014-05-05	2014-05-26
2014-08-25	2014-12-25	2014-12-26	2015-01-01	2015-04-03
2015-04-06	2015-05-04	2015-05-25	2015-08-31	2015-12-25
2016-01-01	2016-03-25	2016-12-25	2016-12-26	2017-04-14
2017-12-25	2018-03-30	2018-12-25	2019-01-01	2019-04-19
2019-04-22	2019-05-06	2019-05-27	2019-08-26	2019-12-25
2019-12-26	2020-01-01	2020-04-10	2020-04-13	2020-05-04
2020-05-25	2020-08-31	2020-12-25	2020-12-26	2020-12-28
2021-01-01	2021-04-02	2021-04-05	2021-05-03	2021-05-31
2021-08-30	2021-12-25	2021-12-26	2021-12-27	2021-12-28
2022-01-03	2022-04-15	2022-04-18	2022-05-02	2022-05-30
2022-08-29	2022-12-25	2022-12-26	2022-12-27	2023-01-02
2023-04-07	2023-04-10	2023-05-01	2023-05-29	2023-08-28
2023-12-25	2023-12-26	2024-01-01	2024-03-29	2024-04-01
2024-05-06	2024-05-27			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UK 100 contract in the CFD market trades in the `Europe/London` time zone.

Market Hours

UK10YBGBP

Introduction

This page shows the trading hours, holidays, and time zone of the UK 10Y Gilt contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UK 10Y Gilt contract in the CFD market:

Weekday	**Time (Europe/London)**
Monday	08:00:00 to 18:00:00
Tuesday	08:00:00 to 18:00:00
Wednesday	08:00:00 to 18:00:00
Thursday	08:00:00 to 18:00:00
Friday	08:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the UK 10Y Gilt contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-04-17	2017-05-01	2017-12-25	2017-12-26
2018-01-01	2018-03-20	2018-04-02	2018-05-01	2018-12-25
2018-12-26	2019-01-01	2019-04-19	2019-04-22	2019-05-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-12-25	2021-01-01	2021-04-02	2021-04-05
2022-04-15	2022-04-18	2022-12-26	2023-01-02	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UK 10Y Gilt contract in the CFD market trades in the `Europe/London` time zone.

Market Hours

US2000USD

Introduction

This page shows the trading hours, holidays, and time zone of the US Russ 2000 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Russ 2000 contract in the CFD market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Russ 2000 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-12-25	2004-01-01	2004-04-09	2004-06-11
2004-12-24	2005-03-25	2006-04-14	2006-12-25	2007-01-01
2007-12-25	2008-01-01	2008-03-21	2008-12-25	2009-01-01
2009-04-10	2009-12-25	2010-01-01	2010-12-24	2011-04-22
2011-12-26	2012-01-02	2012-12-25	2013-01-01	2013-03-29
2013-12-25	2014-01-01	2014-04-18	2014-12-25	2015-01-01
2015-12-25	2016-01-01	2016-03-25	2017-04-14	2018-03-30
2019-01-01	2019-04-19	2019-12-25	2020-01-01	2020-04-10
2020-12-25	2021-01-01	2021-04-02	2021-12-24	2022-01-01
2022-04-15	2022-12-25	2022-12-26	2023-01-01	2023-01-02
2023-04-07	2023-12-25	2024-01-01	2024-03-29	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Russ 2000 contract in the CFD market trades in the `America/New York` time zone.

Market Hours

US30USD

Introduction

This page shows the trading hours, holidays, and time zone of the US Wall St 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Wall St 30 contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Wall St 30 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-05-26	2003-07-04	2003-09-01	2003-11-27
2003-12-25	2004-01-01	2004-01-19	2004-02-16	2004-04-09
2004-05-31	2004-06-11	2004-07-05	2004-09-06	2004-11-25
2004-12-24	2005-01-17	2005-02-21	2005-03-25	2005-05-30
2005-07-04	2005-09-05	2005-11-24	2005-12-26	2006-01-02
2006-01-16	2006-02-20	2006-04-14	2006-11-23	2006-12-25
2007-01-01	2007-01-15	2007-02-19	2007-11-22	2007-12-25
2008-01-01	2008-03-21	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-04-22	2011-12-26
2012-01-02	2012-12-25	2013-01-01	2013-03-29	2013-12-25
2014-01-01	2014-04-18	2015-12-25	2016-03-25	2017-04-14
2018-03-30	2019-01-01	2019-04-19	2019-12-25	2020-01-01
2020-04-10	2020-12-25	2021-01-01	2021-04-02	2021-12-24
2022-01-01	2022-04-15	2022-12-26	2023-01-01	2023-01-02
2023-04-07	2023-12-25	2024-01-01	2024-03-29	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Wall St 30 contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

USB02YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US 2Y T-Note contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 2Y T-Note contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US 2Y T-Note contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 2Y T-Note contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

USB05YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US 5Y T-Note contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 5Y T-Note contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US 5Y T-Note contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 5Y T-Note contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

USB10YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US 10Y T-Note contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 10Y T-Note contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US 10Y T-Note contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 10Y T-Note contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

USB30YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US T-Bond contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US T-Bond contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US T-Bond contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US T-Bond contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

WHEATUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Wheat contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Wheat contract in the CFD market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Tuesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Wednesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Thursday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Friday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Wheat contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24	2022-12-25	2022-12-26
2023-01-01	2023-01-02			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Wheat contract in the CFD market trades in the `America/Chicago` time zone.

Market Hours

WTICOUSD

Introduction

This page shows the trading hours, holidays, and time zone of the West Texas Oil contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the West Texas Oil contract in the CFD market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the West Texas Oil contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-29	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-09-06	2021-11-25	2021-12-24	2022-01-17	2022-02-21
2022-04-15	2022-05-30	2022-07-04	2022-09-05	2022-11-24
2022-12-25	2022-12-26	2023-01-01	2023-01-02	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The West Texas Oil contract in the CFD market trades in the `America/New York` time zone.

Market Hours

XAGAUD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/AUD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/AUD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/AUD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/AUD contract in the CFD market trades in the UTC time zone.

Market Hours

XAGCAD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/CAD contract in the CFD market:

Weekday	Time (UTC)
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/CAD contract in the CFD market:

Date (*yyyy-mm-dd*)			
2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/CAD contract in the CFD market trades in the UTC time zone.

Market Hours

XAGCHF

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/CHF contract in the CFD market:

Weekday	Time (UTC)
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/CHF contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/CHF contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAGEUR

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/EUR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/EUR contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/EUR contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/EUR contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAGGBP

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/GBP contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/GBP contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/GBP contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/GBP contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAGHKD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/HKD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/HKD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/HKD contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAGJPY

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/JPY contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/JPY contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/JPY contract in the CFD market trades in the UTC time zone.

Market Hours

XAGNZD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/NZD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/NZD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/NZD contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAGSGD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/SGD contract in the CFD market:

Weekday	Time (UTC)
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/SGD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/SGD contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAGUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:03:00 to 24:00:00
Monday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Tuesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Wednesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Thursday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Friday	00:00:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver contract in the CFD market trades in the `America/New York` time zone.

Market Hours

XAUAUD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/AUD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/AUD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/AUD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/AUD contract in the CFD market trades in the UTC time zone.

Market Hours

XAUCAD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/CAD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/CAD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/CAD contract in the CFD market trades in the UTC time zone.

Market Hours

XAUCHF

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/CHF contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/CHF contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/CHF contract in the CFD market trades in the UTC time zone.

Market Hours

XAUEUR

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/EUR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/EUR contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/EUR contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/EUR contract in the CFD market trades in the UTC time zone.

Market Hours

XAUGBP

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/GBP contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/GBP contract in the CFD market:

Weekday	Time (UTC)
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/GBP contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/GBP contract in the CFD market trades in the UTC time zone.

Market Hours

XAUHKD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/HKD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/HKD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/HKD contract in the CFD market trades in the UTC time zone.

Market Hours

XAUJPY

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/JPY contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/JPY contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/JPY contract in the CFD market trades in the UTC time zone.

Market Hours

XAUNZD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/NZD contract in the CFD market:

Weekday	Time (UTC)
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/NZD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/NZD contract in the CFD market trades in the UTC time zone.

Market Hours

XAUSGD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/SGD contract in the CFD market:

Weekday	Time (UTC)
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/SGD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/SGD contract in the CFD market trades in the `UTC` time zone.

Market Hours

XAUUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:03:00 to 24:00:00
Monday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Tuesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Wednesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Thursday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Friday	00:00:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold contract in the CFD market trades in the `America/New York` time zone.

Market Hours

XAUXAG

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/Silver contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/Silver contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/Silver contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/Silver contract in the CFD market trades in the `UTC` time zone.

Market Hours

XCUUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Copper contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Copper contract in the CFD market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Copper contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Copper contract in the CFD market trades in the `America/New York` time zone.

Market Hours

XPDUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Palladium contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Palladium contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Palladium contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Palladium contract in the CFD market trades in the `America/New York` time zone.

Market Hours

XPTUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Platinum contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Platinum contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Platinum contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Platinum contract in the CFD market trades in the `America/New York` time zone.

Portfolio

Portfolio

Key Concepts

Introduction

The `Portfolio` object provides information about the whole portfolio state.

Properties

The `Portfolio` object has the following properties:

```py
invested = self.Portfolio.Invested
value = self.Portfolio.TotalPortfolioValue
```

Cost Averaging Accounting

LEAN uses the cost averaging accounting method, which determines the cost of your holdings by taking a weighted average of all your purchase prices. For example, say you place the following buy orders:

1. Buy 10 ABC @ $10
2. Buy 5 ABC @ $11
3. Buy 20 ABC @ $14
4. Buy 3 ABC @ $9

In the preceding example, the average cost of your ABC position is (10*10 + 5*11 + 20*14 + 3*9) / (10 + 5 + 20 + 3) = 12.1579/share. In contrast, if you use the first-in, first-out (FIFO) accounting method, the cost of the first 10 shares is 10/share, not 12.1579/share.

To get the cost of your security holdings, use the `HoldingsCost` property of the SecurityHolding object . If you fill buy and sell orders, the holdings cost is the product of the holding quantity and the average price. For example, the following table shows how the average price and holdings cost changes with each buy and sell order order in a long position:

Order Quantity	Fill Price ($)	Holding Quantity	Average Price ($)	Holdings Cost ($)
2	10	2	(2 * 10) / 2 = 10	2 * 10 = 20
-1	11	1	(1 * 10) / 1 = 10	1 * 10 = 10
1	12	2	(1 * 10 + 1 * 12) / 2 = 11	2 * 11 = 22
-2	13	0	0	0 * 0 = 0

The following table shows how the average price and holdings cost changes with each buy and sell order order in a short position:

Order Quantity	Fill Price ($)	Holding Quantity	Average Price ($)	Holdings Cost ($)
-2	10	-2	(-2 * 10) / -2 = 10	-2 * 10 = -20
1	11	-1	(-1 * 10) / -1 = 10	-1 * 10 = -10
-1	12	-2	(-1 * 10 + (-1) * 12) / -2 = 11	-2 * 11 = -22
2	13	0	0	0 * 0 = 0

Note that when you decrease the position size without closing the trade, the average price doesn't change because the denominator and the coefficients in the numerator of its formula are scaled by the quotient of the current holding quantity and the previous holding quantity. For instance, if the last row in the preceding table had an order quantity 1, the holding quantity would be -1 and the average price would be

$$\frac{(-1 \cdot q) \cdot 10 + (-1 \cdot q) \cdot 12}{-2 \cdot q}$$

$$= \frac{(-1 \cdot \frac{-1}{-2}) \cdot 10 + (-1 \cdot \frac{-1}{-2}) \cdot 12}{-2 \cdot \frac{-1}{-2}}$$

$$= \frac{\frac{-1}{2} \cdot 10 + \frac{-1}{2} \cdot 12}{-1}$$

$$= \frac{-5-6}{-1} = 11$$

Long and Short Holdings

You can identify a position as long- or short-biased based on the sign of the holding quantity. Long positions have a positive quantity and short positions have a negative quantity.

Buying Power

To get the maximum buying power in your account currency you can use for a given `Symbol` and order direction, call the `GetBuyingPower` method.

```
available_buying_power = self.Portfolio.GetBuyingPower(self.symbol, OrderDirection.Buy)
```

PY

For more information about buying power, see Buying Power .

Portfolio

Holdings

Introduction

The `Portfolio` is a dictionary where the key is a `Symbol` and the value is a `SecurityHolding` .

```
security_holding = self.Portfolio["SPY"]
```

PY

Properties

`SecurityHolding` objects have the following properties:

```
security_holding = self.Portfolio["SPY"]
quantity = security_holding.Quantity
invested = security_holding.Invested
```

PY

Get Total Close Profit

To get the profit of a position holding if you closed the position, call the `TotalCloseProfit` method. The value this method returns is denominated in your account currency and accounts for order fees.

```
profit = self.Portfolio["SPY"].TotalCloseProfit()
```

PY

The `TotalCloseProfit` method accepts the following optional arguments:

Argument	Data Type	Description	Default Value
includeFees	bool	Whether to reduce the profit based on the estimated fee from the fee model .	True
exitPrice	float/NoneType	A hypothetical exit price to use for the profit calculation. If you don't provide a value, it uses the bid price for sell orders or the ask price for buy orders.	None
entryPrice	float/NoneType	A hypothetical exit price to use for the profit calculation. If you don't provide a value, it uses the average price of the SecurityHolding .	None
quantity	float/NoneType	The quantity to liquidate. If you don't provide a value, it uses the quantity of the SecurityHolding .	None

LEAN uses this method to define the UnrealizedProfit property.

Get Quantity Value

To get the value of a security at any quantity, call the GetQuantityValue method. The value this method returns is denominated in your account currency.

```
# Get the quantity value at the current price
value_at_current_price = self.Portfolio["SPY"].GetQuantityValue(100)

# Get the quantity value at a specific price
value_at_specific_price = self.Portfolio["SPY"].GetQuantityValue(100, price=30)
```

Portfolio

Cashbook

Introduction

The `CashBook` is a dictionary where the keys are currency tickers and the values are `Cash` objects. The `Cash` objects track the amount of each currency in the portfolio. As you buy and sell securities, LEAN credits and debits the `Cash` objects to reflect your cash holdings.

Account Currency

The default account currency is USD, but you can change it. All of the properties of the Portfolio object that return a currency value denominate the currency value in your account currency. Depending on your account currency and security subscriptions, LEAN may add internal security subscriptions to calculate the `ValueInAccountCurrency` . For example, if you only add BTCETH to your algorithm and set the account currency to USDT, LEAN adds BTCUSDT and ETHUSDT as internal feeds.

To get your account currency, use the `AccountCurrency` property.

```
account_currency = self.AccountCurrency
```

To set your account currency, in the `Initialize` method, call the `SetAccountCurrency` method. You can only set the account currency once. LEAN ignores each additional call to `SetAccountCurrency` .

```
self.SetAccountCurrency("EUR")
```

Settled vs Unsettled Cash

The `Portfolio` has two independent cashbooks. The `CashBook` tracks settled cash and the `UnsettledCashBook` tracks unsettled cash, which you can't spend. If you trade with a margin account, trades settle immediately so LEAN credits and debits the `CashBook` at the time of the trade. In some cases, transactions can take time to settle. For example, Equity trades in a cash account settle in T+3. Therefore, if you sell shares of stock on Monday, the transaction settles on Thursday. In contrast, Option trades settle in T+1.

```
settled_cash_book = self.Portfolio.CashBook
unsettled_cash_book = self.Portfolio.UnsettledCashBook
```

Track Cash Balances

To get the balance of a currency in the cash book, use the `Amount` property.

```
usd = self.Portfolio.CashBook["USD"].Amount
btc = self.Portfolio.CashBook["BTC"].Amount
```

To get the value of a currency in the cash book, denominated in your account currency, use the `ValueInAccountCurrency` property.

```
eth_value = self.Portfolio.CashBook["ETH"].ValueInAccountCurrency
```

Deposits and Withdraws

In backtests, you can add and remove cash from the cash book. To add or remove cash, call the `AddAmount` method.

```
new_usd_balance = self.Portfolio.CashBook["USD"].AddAmount(100)
new_btc_balance = self.Portfolio.CashBook["BTC"].AddAmount(-1.5)
```

In live trading, add and withdraw cash through your brokerage account. If you adjust the cash balances in your algorithm with the `AddAmount` method, the cash balance in your algorithm will be out of sync with the cash balance in your brokerage account.

Currency Symbols

A currency symbol is a graphic that represents the currency name. For example, $ is for dollars, € for euros, and ฿ for Bitcoin. To get the symbol of a currency in the cash book, use `CurrencySymbol` property.

```
usd_symbol = self.Portfolio.CashBook["USD"].CurrencySymbol
```

Conversion Rates

To get the conversion rate for a currency in the cashbook to your account currency, use the `ConversionRate` property.

```
eur_conversion_rate = self.Portfolio.CashBook["EUR"].ConversionRate
```

Universes

Universes

Key Concepts

Introduction

Universe selection is the process of selecting a basket of assets you may trade. Dynamic universe selection increase diversification and decrease selection bias in your algorithm.

How Universe Selection Works

When you add a universe to your algorithm, LEAN sends a large dataset into a filter function you define. Your filter function needs to return a list of `Symbol` objects. LEAN automatically subscribes to these new symbols and adds them to your algorithm. Your algorithm can do almost anything inside your filter functions, but the goal should be to narrow down the set of securities to the securities that are most applicable for your algorithm.

Security Changed Events

When your universe adds and removes assets, LEAN notifies your algorithm through the `OnSecuritiesChanged` event handler. The event handler receives a `SecurityChanges` object, which contains references to the added and removed securities. To access the added securities, check the `changes.AddedSecurities` property. To access the removed securities, check the `changes.RemovedSecurities` property.

```py
def OnSecuritiesChanged(self, changes: SecurityChanges) -> None:
    for security in changes.AddedSecurities:
        self.Debug(f"{self.Time}: Added {security.Symbol}")

    for security in changes.RemovedSecurities:
        self.Debug(f"{self.Time}: Removed {security.Symbol}")

        if security.Invested:
            self.Liquidate(security.Symbol, "Removed from Universe")
```

The preceding example liquidates securities that leave the universe. In this case, LEAN creates a market on open order and frees up buying power when the market opens.

A convenient way to track the securities that are currently in the universe is to maintain a `self.securities` list.

```
# In Initialize
self.securities = []

def OnSecuritiesChanged(self, changes: SecurityChanges) -> None:
    for security in changes.RemovedSecurities:
        if security in self.securities:
            self.securities.remove(security)

    self.securities.extend(changes.AddedSecurities)
```

If you need to save data or create objects for each security in the universe, maintain a dictionary of custom `SymbolData` objects. This technique is useful if you want to track stop loss levels or add indicators for each asset in the universe.

```
# In Initialize
self.symbol_data_by_symbol = {}

def OnSecuritiesChanged(self, changes: SecurityChanges) -> None:
    for security in changes.AddedSecurities:
        self.symbol_data_by_symbol[security.Symbol] = SymbolData() # You need to define this class

    for security in changes.RemovedSecurities:
        self.symbol_data_by_symbol.pop(security.Symbol, None)
```

Select Current Constituents

If you don't want to make any changes to the current universe, return `Universe.Unchanged` from your filter functions.

```
class MyUniverseAlgorithm(QCAlgorithm):
     def Initialize(self) -> None:
         self.AddUniverse(self.MyCoarseFilterFunction)

    def MyCoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
         return Universe.Unchanged
```

Universe Manager

The universe manager tracks all the universes in your algorithm. If you add multiple universe, you can access the constituents of each individual universe. To access the constituents of a universe in a multi-universe algorithm, save references to each universe when you add them.

```
# In Initialize
self.universe = self.AddUniverse(self.MyCoarseFilterFunction)

# In OnData
universe_members = self.UniverseManager[self.universe.Configuration.Symbol].Members
for kvp in universe_members:
    symbol = kvp.Key
    security = kvp.Value
```

Active Securities

The `ActiveSecurities` property of the algorithm class contains all of the assets currently in your universe. It is a dictionary where the key is a `Symbol` and the value is a `Security` . When you remove an asset from a universe, LEAN usually removes the security from the `ActiveSecurities` collection and removes the security subscription. However, it

won't remove the security in any of the following situations:

- You own the security.
- You have an open order for the security.
- The security wasn't in the universe long enough to meet the `MinimumTimeInUniverse` setting.

When LEAN removes the security, the `Security` object remains in the `Securities` collection for record-keeping purposes, like tracking fees and trading volume.

Derivative Universes

In a regular universe, you select a basket of assets from the entire universe of securities. In a derivative universe, you select a basket of contracts for an underlying asset. The following derivative universes are available:

- Equity Options
- Futures
- Future Options
- Index Options

Universes

Settings

Introduction

Universe settings and security initializers enable you to configure some properties of the securities in a universe.

Properties

The universe settings of your algorithm configure some properties of the universe constituents. The following table describes the properties of the `UniverseSettings` object:

Property: `ExtendedMarketHours` Should assets also feed extended market hours? You only receive extended market hours data if you create the subscription with an intraday resolution. If you create the subscription with daily resolution, the daily bars only reflect the regular trading hours. Data Type: `bool` | Default Value: `False`
Property: `FillForward` Should asset data fill forward? Data Type: `bool` | Default Value: `True`
Property: `MinimumTimeInUniverse` What's the minimum time assets should be in the universe? Data Type: `timedelta` | Default Value: `timedelta(1)`
Property: `Resolution` What resolution should assets use? Data Type: `Resolution` | Default Value: `Resolution.Minute`

Property: `ContractDepthOffset` What offset from the current front month should be used for continuous Future contracts ? 0 uses the front month and 1 uses the back month contract. This setting is only available for Future assets. Data Type: `int` | Default Value: `0`
Property: `DataMappingMode` How should continuous Future contracts be mapped? This setting is only available for Future assets. Data Type: `DataMappingMode` | Default Value: `DataMappingMode.OpenInterest`
Property: `DataNormalizationMode` How should historical prices be adjusted? This setting is only available for Equity and Futures assets. Data Type: `DataNormalizationMode` | Default Value: `DataNormalizationMode.Adjusted`
Property: `Leverage` What leverage should assets use in the universe? This setting is not available for derivative assets. Data Type: `float` | Default Value: `Security.NullLeverage`

To set the `UniverseSettings` , update the preceding properties in the `Initialize` method before you add the "universe" These settings are globals, so they apply to all universes you create.

```py
# Request second resolution data. This will be slow!
self.UniverseSettings.Resolution = Resolution.Second
self.AddUniverse(self.MyCoarseFilterFunction)
```

Configure Universe Securities

Instead of configuring global universe settings, you can individually configure the settings of each security in the universe with a security initializer. Security initializers let you apply any security-level reality model or special data requests on a per-security basis. To set the security initializer, in the `Initialize` method, call the `SetSecurityInitializer` method and then define the security initializer.

```
#In Initialize
self.SetSecurityInitializer(self.CustomSecurityInitializer)

def CustomSecurityInitializer(self, security: Security) -> None:
    # Disable trading fees
    security.SetFeeModel(ConstantFeeModel(0, "USD"))
```

For simple requests, you can use the functional implementation of the security initializer. This style lets you configure the security object with one line of code.

```
self.SetSecurityInitializer(lambda security: security.SetFeeModel(ConstantFeeModel(0, "USD")))
```

In some cases, you may want to trade a security in the same time loop that you create the security subscription. To avoid errors, use a security initializer to set the market price of each security to the last known price. The `GetLastKnownPrices` method seeds the security price by gathering the security data over the last 3 days. If there is no data during this period, the security price remains at 0.

```
seeder = FuncSecuritySeeder(self.GetLastKnownPrices)
self.SetSecurityInitializer(lambda security: seeder.SeedSecurity(security))
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage
- Fee
- Buying Power
- Settlement
- Margin Interest Rate

The default security initializer also sets the leverage of each security and intializes each security with a seeder function. To extend upon the default security initializer instead of overwriting it, create a custom `BrokerageModelSecurityInitializer` .

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetFeeModel(ConstantFeeModel(0, "USD"))
```

To set a seeder function without overwriting the reality models of the brokerage, use the standard `BrokerageModelSecurityInitializer`.

```
PY
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))
```

Selection Frequency

Most universes run on a daily basis, but some can select assets at different frequencies.

Universes

Equity

Introduction

There are several ways to create an Equities universe. You can select a universe based on `CoarseFundamental` data or the constituents of an ETF, and then you can further filter your universe down with corporate fundamentals. The following sections explain each of these techniques in detail.

Coarse Universe Selection

A coarse universe enables you pick a set of stocks based on their trading volume, price, or whether they have fundamental data. To add a coarse universe, in the `Initialize` method, pass a filter function to the `AddUniverse` method. The coarse filter function receives a list of `CoarseFundamental` objects and must return a list of `Symbol` objects. The `Symbol` objects you return from the function are the constituents of the universe and LEAN automatically creates subscriptions for them. Don't call `AddEquity` in the filter function.

```
PY
class MyCoarseUniverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.AddUniverse(self.CoarseFilterFunction)

    def CoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        sorted_by_dollar_volume = sorted(coarse, key=lambda x: x.DollarVolume, reverse=True)
        return [c.Symbol for c in sorted_by_dollar_volume[:100]]
```

`CoarseFundamental` objects have the following attributes:

The total number of stocks in the US Equity Security Master dataset is 30,000 but your coarse filter function won't receive all of these at one time because the US Equity Security Master dataset is free of survivorship bias and some of the securities have delisted over time. The number of securities that are passed into your coarse filter function depends on the date of your algorithm. Currently, there are about 10,000 securities that LEAN passes into your coarse filter function.

Dollar Volume Selection

A dollar volume universe enables you pick the most liquid stocks in the market with just one line of code. To add a dollar volume universe, call the `Universe.DollarVolume.Top` helper method and pass the result to the `AddUniverse` method.

```
PY
// Add the 50 stocks with the highest dollar volume
self.AddUniverse(self.Universe.DollarVolume.Top(50))
```

ETF Constituents Selection

An ETF constituents universe lets you select a universe of securities in an ETF. The US ETF Constituents dataset

includes 2,650 US ETFs you can use to create your universe. To add an ETF Constituents universe, call the `Universe.ETF` method.

```
class ETFConstituentsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.AddUniverse(self.Universe.ETF("SPY", Market.USA, self.UniverseSettings))
```

The following table describes the `ETF` method arguments:

Argument: `etfTicker` The ETF ticker. To view the supported ETFs in the US ETF Constituents dataset, see Supported ETFs . Data Type: `str` | Default Value: None
Argument: `market` The market of the ETF. If you don't provide an argument, it uses the default Equity market of the brokerage model . Data Type: `str` | Default Value: None
Argument: `universeSettings` The universe settings . If you don't provide an argument, it uses the algorithm `UniverseSettings` . Data Type: `UniverseSettings` | Default Value: None
Argument: `universeFilterFunc` A function to select some of the ETF constituents for the universe. If you don't provide an argument, it selects all of the constituents. Data Type: `Callable[[List[ETFConstituentData]], List[Symbol]]` | Default Value: `None`

To select a subset of the ETF constituents, provide a `universeFilterFunc` argument. The filter function receives `ETFConstituentData` objects, which represent one of the ETF constituents. `ETFConstituentsData` objects have the following attributes:

```py
class ETFConstituentsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        universe = self.Universe.ETF("SPY", Market.USA, self.UniverseSettings, self.ETFConstituentsFilter)
        self.AddUniverse(universe)

    def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
        # Get the 10 securities with the largest weight in the index
        selected = sorted([c for c in constituents if c.Weight],
            key=lambda c: c.Weight, reverse=True)[:10]
        return [c.Symbol for c in selected]
```

Fundamentals Selection

A fundamental universe lets you select stocks based on corporate fundamental data. This data is powered by Morningstar® and includes approximately 5,000 tickers with 900 properties each. Due to the sheer volume of information, fundamental selection is performed on the output of another universe filter. Think of this process as a 2-stage filter. An initial filter function selects a set of stocks and then a fine fundamental filter function selects a subset of those stocks.

QuantConnect Coarse and Fine Universe Selection

To add a fundamental universe, in the `Initialize` method, pass two filter functions to the `AddUniverse` method. The first filter function can be a coarse universe filter , dollar volume filter , or an ETF constituents filter . The second filter function receives a list of `FineFundamental` objects and must return a list of `Symbol` objects. The list of `FineFundamental` objects contains a subset of the `Symbol` objects that the first filter function returned. The `Symbol` objects you return from the second function are the constituents of the fundamental universe and LEAN automatically creates subscriptions for them. Don't call `AddEquity` in the filter function.

> **Tip:** Only 5,000 assets have fundamental data. If your first filter function receives `CoarseFundamental` data, you should only select assets that have a true value for their `HasFundamentalData` property.

```py
class MyUniverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.AddUniverse(self.CoarseFilterFunction, self.FineFundamentalFunction)

    def CoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        # In addition to further coarse universe selection, ensure the security has fundamental data
        return [c.Symbol for c in coarse if c.HasFundamentalData]

    def FineFundamentalFunction(self, fine: List[FineFundamental]) -> List[Symbol]:
        # Return a list of Symbols
```

`FineFundamental` objects have the following attributes:

Example

The simplest example of accessing the fundamental object would be harnessing the iconic PE ratio for a stock. This is a ratio of the price it commands to the earnings of a stock. The lower the PE ratio for a stock, the more affordable it appears.

```
PY
# In Initialize:
self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)

def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    sortedByDollarVolume = sorted(coarse, key=lambda x: x.DollarVolume, reverse=True)
    filtered = [x.Symbol for x in sortedByDollarVolume if x.HasFundamentalData]
    return filtered[:50]

def FineSelectionFunction(self, fine: List[FineFundamental]) -> List[Symbol]:
    sortedByPeRatio = sorted(fine, key=lambda x: x.ValuationRatios.PERatio, reverse=False)
    return [x.Symbol for x in sortedByPeRatio[:10]]
```

Asset Categories

In addition to valuation ratios, the US Fundamental Data from Morningstar has many other data point attributes, including over 200 different categorization fields for each US stock. Morningstar groups these fields into sectors, industry groups, and industries.

Sectors are large super categories of data. To get the sector of a stock, use the `MorningstarSectorCode` property.

```
PY
tech = [x for x in fine if x.AssetClassification.MorningstarSectorCode ==
MorningstarSectorCode.Technology]
```

Industry groups are clusters of related industries that tie together. To get the industry group of a stock, use the `MorningstarIndustryGroupCode` property.

```
PY
ag = [x for x in fine if x.AssetClassification.MorningstarIndustryGroupCode ==
MorningstarIndustryGroupCode.Agriculture]
```

Industries are the finest level of classification available. They are the individual industries according to the Morningstar classification system. To get the industry of a stock, use the `MorningstarIndustryCode` .

```
PY
coal = [x for x in fine if x.AssetClassification.MorningstarIndustryCode == MorningstarSectorCode.Coal]
```

Practical Limitations

Like coarse universes, fine universes allow you to select an unlimited universe of assets to analyze. Each asset in the universe consumes approximately 5MB of RAM, so you may quickly run out of memory if your universe filter selects many assets. If you backtest your algorithms in the Algorithm Lab, familiarize yourself with the RAM capacity of your backtesting and live trading nodes . To keep your algorithm fast and efficient, only subscribe to the assets you need.

Selection Frequency

Equity universes run on a daily basis.

Live Trading Considerations

The live data for coarse and fine universe selection arrives at 7 AM Eastern Time (ET), so coarse and fine universe selection runs for live algorithms between 7 and 8 AM ET. This timing allows you to place trades before the market opens. Don't schedule anything for midnight because the universe selection data isn't ready yet.

Examples

The following examples are typical filter functions you may want.

Example 1: Take 500 stocks that are worth more than 10*andhavemorethan*10M daily trading volume

The most common use case is to select a lot of liquid stocks. With a coarse universe filter, this is simple and fast. The following example selects the top most liquid 500 stocks over $10 per share.

```
PY
def CoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    sortedByDollarVolume = sorted(coarse, key=lambda x: x.DollarVolume, reverse=True)
    filtered = [ x.Symbol for x in sortedByDollarVolume
                 if x.Price > 10 and x.DollarVolume > 10000000 ]
    return filtered[:500]
```

Example 2: Take 10 stocks above their 200-Day EMA and have more than $1B daily trading volume

Another common request is to filter the universe by a technical indicator, such as only picking stocks above their 200-day EMA. The `CoarseFundamental` object has adjusted price and volume information, so you can do any price-related analysis.

```
PY
# setup state storage in initialize method
self.stateData = { }

def CoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    # We are going to use a dictionary to refer the object that will keep the moving averages
    for c in coarse:
        if c.Symbol not in self.stateData:
            self.stateData[c.Symbol] = SelectionData(c.Symbol, 200)

        # Updates the SymbolData object with current EOD price
        avg = self.stateData[c.Symbol]
        avg.update(c.EndTime, c.AdjustedPrice, c.DollarVolume)

    # Filter the values of the dict to those above EMA and more than $1B vol.
    values = [x for x in self.stateData.values() if x.is_above_ema and x.volume > 1000000000]

    # sort by the largest in volume.
    values.sort(key=lambda x: x.volume, reverse=True)

    # we need to return only the symbol objects
    return [ x.symbol for x in values[:10] ]
```

In this example, the `SelectionData` class group variables for the universe selection and updates the indicator of each asset. We highly recommend you follow this pattern to keep your algorithm tidy and bug free. The following snippet shows an example implementation of the `SelectionData` class, but you can make this whatever you need to store your custom universe filters.

```
class SelectionData(object):
    def __init__(self, symbol, period):
        self.symbol = symbol
        self.ema = ExponentialMovingAverage(period)
        self.is_above_ema = False
        self.volume = 0

    def update(self, time, price, volume):
        self.volume = volume
        if self.ema.Update(time, price):
            self.is_above_ema = price > ema
```

Note that the preceding `SelectionData` class uses a manual EMA indicator instead of the automatic version . For more information about universes that select assets based on indicators, see Indicator Universes .

Example 3: Take 10 stocks that are the furthest above their 10-day SMA of volume

The process to get the 10-day SMA stock volume is the same process as in Example 2. First, you should define a `SelectionData` class that performs the averaging. For this example, the following class will serve this purpose:

```
class SelectionData(object):
    def __init__(self, symbol, period):
        self.symbol = symbol
        self.volume = 0
        self.volume_ratio = 0
        self.sma = SimpleMovingAverage(period)

    def update(self, time, price, volume):
        self.volume = volume
        if self.sma.Update(time, volume):
            # get ratio of this volume bar vs previous 10 before it.
            self.volume_ratio = volume / self.sma.Current.Value
```

This class tracks the ratio of today's volume relative to historical volumes. You can use this ratio to select assets that are above their 10-day simple moving average and sort the results by the ones that have had the biggest jump since yesterday.

```
def CoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    for c in coarse:
        if c.Symbol not in self.stateData:
            self.stateData[c.Symbol] = SelectionData(c.Symbol, 10)
        avg = self.stateData[c.Symbol]
        avg.update(c.EndTime, c.AdjustedPrice, c.DollarVolume)

    # filter the values of selectionData(sd) above SMA
    values = [sd for sd in self.stateData.values() if sd.volume > sd.sma.Current.Value and sd.volume_ratio
> 0]

    # sort sd by the largest % jump in volume.
    values.sort(key=lambda sd: sd.volume_ratio, reverse=True)

    # return the top 10 symbol objects
    return [ sd.symbol for sd in values[:10] ]
```

Example 4: Take the top 10 "fastest moving" stocks with a 50-Day EMA > 200 Day EMA

You can construct complex universe filters with the `SelectionData` helper class pattern. To view a full example of this algorithm, see the EmaCrossUniverseSelectionAlgorithm in the LEAN GitHub repository or take the related Boot Camp lesson .

Demonstration Algorithms
CoarseUniverseTop3DollarVolumeAlgorithm.py Python EmaCrossUniverseSelectionAlgorithm.py Python
DropboxUniverseSelectionAlgorithm.py Python WeeklyUniverseSelectionRegressionAlgorithm.py Python
CoarseFineFundamentalComboAlgorithm.py Python

Universes

Equity Options

Introduction

An Equity Options universe lets you select a basket of contracts for a single Option. LEAN models Option subscriptions as a universe of Option contracts.

Create Universes

To add a universe of Equity Option contracts, in the `Initialize` method, call the `AddOption` method. This method returns an `Option` object, which contains the canonical `Symbol` . You can't trade with the canonical Option `Symbol` , but save a reference to it so you can easily access the Option contracts in the OptionChain that LEAN passes to the `OnData` method.

```
PY
option = self.AddOption("SPY")
self.symbol = option.Symbol
```

The following table describes the `AddOption` method arguments:

Argument	Data Type	Description	Default Value
`ticker`	`str`	The underlying Equity ticker. To view the supported underlying Equity tickers, see Supported Assets .	
`resolution`	`Resolution/NoneType`	The resolution of the market data. To view the supported resolutions, see Resolutions . The Equity resolution must be less than or equal to the Equity Option resolution. For example, if you set the Equity resolution to minute, then you must set the Equity Option resolution to minute, hour, or daily.	`None`
`market`	`str`	The underlying Equity market.	`None`
`fillForward`	`bool`	If true, the current slice contains the last available data even if there is no data at the current time.	`True`
`leverage`	`float`	The leverage for this Equity.	`Security.NullLeverage`
`extendedMarketHours`	`bool`	A flag that signals if LEAN should send data during pre- and post-market trading hours.	`False`

If you add an Equity Option universe but don't have a subscription to the underlying Equity, LEAN automatically subscribes to the underlying Equity with the following settings:

Setting	Value
Fill forward	Same as the Option universe
Leverage	0
Extended Market Hours	Same as the Option universe
Data Normalization	`DataNormalizationMode.Raw`

If you already have a subscription to the underlying Equity but it's not `Raw` data normalization, LEAN automatically changes it to `Raw` .

To override the default pricing model of the Option, set a pricing model .

```
option.PriceModel = OptionPriceModels.CrankNicolsonFD()
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Filter Contracts

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 31 days

To adjust the universe of contracts, set a filter. The filter usually runs at every time step in your algorithm. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Option` object. The following table describes the available filter techniques:

Method	Description
`SetFilter(minStrike: int, maxStrike: int)`	Selects the contracts that have a strike price within a minimum and maximum strike level relative to the underlying price. For example, say the underlying price is 302*andtherearestrikesatevery*5. If you set `minStrike` to -1 and `maxStrike` to 1, LEAN selects the contracts that have a strike of 300*or*305.
`SetFilter(minExpiry: timedelta, maxExpiry: timedelta)`	Selects the contracts that expire within the range you set.
`SetFilter(minStrike: int, maxStrike: int, minExpiry: timedelta, maxExpiry: timedelta)`	Selects the contracts that expire and have a strike within the range you set.
`SetFilter(universeFunc: Callable[[OptionFilterUniverse], OptionFilterUniverse])`	Selects the contracts that a function selects.

```
# Select contracts that have a strike price within 1 strike level above and below the underlying price
option.SetFilter(minStrike=-1, maxStrike=1)

# Select contracts that expire within 30 days
option.SetFilter(minExpiry=timedelta(days=0), maxExpiry=timedelta(days=30))

# Select contracts that have a strike price within 1 strike level and expire within 30 days
option.SetFilter(minStrike=-1, maxStrike=1, minExpiry=timedelta(days=0), maxExpiry=timedelta(days=30))

# Select call contracts
option.SetFilter(lambda option_filter_universe: option_filter_universe.CallsOnly())
```

The following table describes the filter methods of the `OptionFilterUniverse` class:

Method	Description
`Strikes(minStrike: int, maxStrike: int)`	Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price
`CallsOnly()`	Selects call contracts
`PutsOnly()`	Selects put contracts
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weeklys contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```py
# Example 1: Select the front month call contracts
option.SetFilter(lambda option_filter_universe: option_filter_universe.CallsOnly().FrontMonth())

# Example 2: Select the contracts (including weeklys) that expire in the next 90 days
option.SetFilter(lambda option_filter_universe: option_filter_universe.IncludeWeeklys().Strikes(-20,
20).Expiration(0, 90))
```

To perform thorough filtering on the `OptionFilterUniverse` , define an isolated filter method.

```
# In Initialize
option.SetFilter(self.contract_selector)

def contract_selector(self, option_filter_universe: OptionFilterUniverse) -> OptionFilterUniverse:
    symbols = option_filter_universe.PutsOnly()
    strike = min([symbol.ID.StrikePrice for symbol in symbols])
    symbols = [symbol for symbol in symbols if symbol.ID.StrikePrice == strike]
    return option_filter_universe.Contracts(symbols)
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

Navigate Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security. They have the following properties:

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` . After you get the `OptionChain` , you can sort and filter the Option contracts in the chain.

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.symbol)
    if chain:
        # Example: Find 5 put contracts that are closest to at-the-money (ATM) and have the farthest
expiration
        contracts = [x for x in chain if x.Right == OptionRight.Put]
        contracts = sorted(sorted(contracts, \
            key = lambda x: abs(chain.Underlying.Price - x.Strike)), \
            key = lambda x: x.Expiry, reverse=True)[:5]

        # Select the contract with the delta closest to -0.5
        contract = sorted(contracts, key=lambda x: abs(-0.5 - x.Greeks.Delta))[0]
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.OptionChains.items():
        contracts = chain.Contracts
```

Selection Frequency

By default, Equity Option universes run at every time step to select their contracts. If you add the `OnlyApplyFilterAtMarketOpen` method to your contract filter, the universe selects contracts once a day at the first time step.

Live Trading Considerations

By default, LEAN adds contracts to the `OptionChain` that pass the filter criteria at every time step in your algorithm. If a contract has been in the universe for a duration that matches the `MinimumTimeInUniverse` setting and it no longer passes the filter criteria, LEAN removes it from the chain.

This default behavior can lead to a large number of open data subscriptions, which can be inconvenient in live trading

since some data feed providers impose data subscription limits. To reduce the number of data subscriptions, change the value of the `MinimumTimeInUniverse` universe setting .

```
self.UniverseSettings.MinimumTimeInUniverse = timedelta(minutes=15)
self.UniverseSettings.MinimumTimeInUniverse = timedelta(0)
```

If you set the value to `timedelta(0)` , LEAN removes all of the contract subscriptions at every time step and your data provider stops sending the data. If your filter re-selects some of the same contracts when it runs next, LEAN requests new data subscriptions from your data provider. To avoid unsubscribing and resubscribing to the same contracts at every time step, use a `timedelta` greater than 0.

Universes

Crypto

Introduction

A Crypto universe lets you select a basket of Cryptocurrencies based on `CryptoCoarseFundamental` data.

Crypto Universes

To add a universe of Cryptocurrencies, in the `Initialize` method, pass a `CryptoCoarseFundamentalUniverse` to the `AddUniverse` method.

```
def Initialize(self) -> None:
    self.UniverseSettings.Resolution = Resolution.Daily
    self.SetBrokerageModel(BrokerageName.GDAX, AccountType.Cash)

    # Add universe selection of cryptos based on coarse fundamentals
    filter_function = lambda crypto_coarse: [c.Symbol for c in crypto_coarse]
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.GDAX, self.UniverseSettings, filter_function))
```

PY

The following table describes the `CryptoCoarseFundamentalUniverse` constructor arguments:

Argument	Data Type	Description	Default Value
`market`	`str`	The market of the Cryptocurrencies. To view the available Crypto markets, see the CoinAPI datasets .	
`universeSettings`	`UniverseSettings`	The universe settings .	
`universeFilterFunc`	`Callable[[List[CryptoCoarseFundamental]], List[Symbol]]`	A function to select some of the Cryptocurrencies for the universe.	

The filter function receives `CryptoCoarseFundamental` objects, which represent one of the Cryptocurrencies in the market. The `Symbol` objects that the filter function returns represent the universe constituents. `CryptoCoarseFundamental` objects have the following attributes:

To perform thorough filtering on the `CryptoCoarseFundamental` objects, define an isolated filter method.

```py
def Initialize(self) -> None:
    self.UniverseSettings.Resolution = Resolution.Daily
    self.SetBrokerageModel(BrokerageName.GDAX, AccountType.Cash)

    # Add universe selection of cryptos based on coarse fundamentals
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.GDAX, self.UniverseSettings,
self.universe_filter))

def universe_filter(self, crypto_coarse: List[CryptoCoarseFundamental]) -> List[Symbol]:
    # Define the universe selection function
    return [cf.Symbol for cf in crypto_coarse if cf.Volume >= 100 and cf.VolumeInUsd > 10000]
```

Selection Frequency

Crypto universes run on a daily basis.

Live Trading Considerations

In live mode, the pipeline has a 16-hour delay. Your algorithm receives the `CryptoCoarseFundamental` objects at around 16:00-16:30 Coordinated Universal Time (UTC) each day, depending on the data processing task.

Universes

Futures

Introduction

A Futures universe lets you select a basket of contracts for a single Future. LEAN models Future subscriptions as a universe of Future contracts. A Future universe is similar to an Option universe , except Future contracts don't have a strike price, so the universe filter primarily focuses on the contract expiration date.

Create Universes

To add a universe of Future contracts, in the `Initialize` method, call the `AddFuture` method. This method returns an `Future` object, which contains the continuous contract `Symbol` . The `continuous contract Symbol` is the key to access the contracts in the FutureChain that LEAN passes to the `OnData` method. When you create the Future subscription, save a reference to the `continuous contract Symbol` so you can use it later in your algorithm.

```
self.future = self.AddFuture(Futures.Currencies.BTC)
self.symbol = self.future.Symbol
```

The following table describes the `AddFuture` method arguments:

Argument: `ticker` The Future ticker. To view the supported assets in the US Futures dataset, see Supported Assets . Data Type: `str` | Default Value: None
Argument: `resolution` The resolution of the market data. To view the supported resolutions, see Resolutions . If you don't provide a value, it uses `Resolution.Minute` by default. Data Type: `Resolution/NoneType` | Default Value: `None`
Argument: `market` The Futures market. To view the supported markets in the US Futures dataset, see Supported Markets . If you don't provide a value, it uses the default Future market of your brokerage model . Data Type: `str` | Default Value: `None`

Argument: `fillForward`

If true, the current slice contains the last available data even if there is no data at the current time.

Data Type: `bool` | Default Value: `True`

Argument: `leverage`

The leverage for this Future.

Data Type: `float` | Default Value: `Security.NullLeverage`

Argument: `extendedMarketHours`

If true, use data from the pre and post market sessions

Data Type: `bool` | Default Value: `False`

Argument: `dataMappingMode`

The contract mapping mode to use for the continuous future contract

Data Type: `DataMappingMode/NoneType` | Default Value: `None`

Argument: `dataNormalizationMode`

The price scaling mode to use for the continuous future contract

Data Type: `DataNormalizationMode/NoneType` | Default Value: `None`

Argument: `contractDepthOffset`

The continuous future contract desired offset from the current front month. For example, 0 is the front month, 1 is the back month contract.

Data Type: `int` | Default Value: `0`

Continous Contracts

By default, LEAN only subscribes to the continuous Future contract. A continuous Future contract represents a series of separate contracts stitched together to form a continuous price. If you need a lot of historical data to warm up an indicator, apply the indicator to the continuous contract price series. The `Future` object has a `Symbol` property and a `Mapped` property. The price of the `Symbol` property is the adjusted price of the continuous contract. The price of the

`Mapped` property is the raw price of the currently selected contract in the continuous contract series.

```
# Get the adjusted price of the continuous contract
adjusted_price = self.Securities[self.future.Symbol].Price

# Get the raw price of the currently selected contract in the continuous contract series
raw_price = self.Securities[self.future.Mapped].Price
```

To configure how LEAN identifies the current Future contract in the continuous series and how it forms the adjusted price between each contract, provide `dataMappingMode` , `dataNormalizationMode` , and `contractDepthOffset` arguments to the `AddFuture` method. The `Future` object that the `AddFuture` method returns contains a `Mapped` property that references the current contract in the continuous contract series. As the contracts roll over, the `Mapped` property references the next contract in the series and you receive a `SymbolChangedEvent` object in the `OnData` method. The `SymbolChangedEvent` references the old contract `Symbol` and the new contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    for changed_event in slice.SymbolChangedEvents.Values:
        self.Log(f"Contract rollover from {changed_event.OldSymbol} to {changed_event.NewSymbol}")
```

Data Normalization Modes

The `dataNormalizationMode` argument defines how the price series of two contracts are stitched together when the contract rollovers occur. The following `DataNormalizatoinMode` enumeration members are available for continuous contracts:

We use the entire Futures history to adjust historical prices. This process ensures you get the same adjusted prices, regardless of the backtest end date.

Data Mapping Modes

The `dataMappingMode` argument defines when contract rollovers occur. The `DataMappingMode` enumeration has the following members:

Contract Depth Offsets

The `contractDepthOffset` argument defines which contract to use. 0 is the front month contract, 1 is the following back month contract, and 3 is the second back month contract.

Filter Contracts

By default, LEAN doesn't add any contracts to the FuturesChain it passes to the `OnData` method. To add a universe of Future contracts, in the `Initialize` method, call the `SetFilter` method of the `Future` object. The following table describes the available filter techniques:

Method	Description
`SetFilter(minExpiryDays: int, maxExpiryDays: int)`	Selects the contracts that expire within the range you set.
`SetFilter(universeFunc: Callable[[FutureFilterUniverse], FutureFilterUniverse])`	Selects the contracts that a function selects.

```py
# Select the contracts which expire within 182 days
self.future.SetFilter(0, 182)

# Select the front month contract
self.future.SetFilter(lambda future_filter_universe: future_filter_universe.FrontMonth())
```

The following table describes the filter methods of the `FutureFilterUniverse` class:

Method	Description
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weekly contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

The preceding methods return an `FutureFilterUniverse` , so you can chain the methods together.

```py
# Select the front month standard contracts
self.future.SetFilter(lambda future_filter_universe: future_filter_universe.StandardsOnly().FrontMonth())
```

You can also define an isolated filter method.

```
# In Initialize
self.future.SetFilter(self.contract_selector)

def contract_selector(self,
    future_filter_universe: Callable[[FutureFilterUniverse], FutureFilterUniverse]) ->
FutureFilterUniverse:
    return future_filter_universe.StandardsOnly().FrontMonth()
```

Some of the preceding filter methods only set an internal enumeration in the `FutureFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `FutureFilterUniverse` .

By default, LEAN adds contracts to the `FutureChain` that pass the filter criteria at every time step in your algorithm. If a contract has been in the universe for a duration that matches the `MinimumTimeInUniverse` setting and it no longer passes the filter criteria, LEAN removes it from the chain

Navigate Futures Chains

`FuturesChain` objects represent an entire chain of contracts for a single underlying Future. They have the following properties:

To get the `FuturesChain` , index the `FuturesChains` property of the `Slice` with the continuous contract `Symbol` .

```
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.symbol)
    if chain:
        # Example: Select the contract with the greatest open interest
        contract = sorted(chain, key=lambda contract: contract.OpenInterest, reverse=True)[0]
```

You can also loop through the `FuturesChains` property to get each `FuturesChain` .

```
def OnData(self, slice: Slice) -> None:
    for continuous_contract_symbol, chain in slice.FuturesChains.items():
        pass
```

Selection Frequency

By default, Futures universes run at every time step to select their contracts. If you add the `OnlyApplyFilterAtMarketOpen` method to your contract filter, the universe selects contracts once a day at the first time step.

Universes

Future Options

Introduction

A Future Option universe lets you select a basket of Option contracts on the contracts in a Futures universe .

Create Universes

To add a universe of Future Option contracts, in the `Initialize` method, define a Future universe and then pass the canonical `Symbol` to the `AddFutureOption` method.

```
future = self.AddFuture(Futures.Metals.Gold)
future.SetFilter(0, 90)
self.AddFutureOption(future.Symbol)
```

PY

The following table describes the `AddFutureOption` method arguments:

Argument	Data Type	Description	Default Value
`symbol`	`Symbol`	The continuous Future contract Symbol. To view the supported assets in the US Future Options dataset, see Supported Assets .	
`optionFilter`	`Callable[[OptionFilterUniverse], OptionFilterUniverse]`	A function that selects Future Option contracts	`None`

To override the default pricing model of the Option, set a pricing model in a security initializer.

```
# In Initialize
seeder = SecuritySeeder.Null
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel, seeder, self))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder, algorithm:
QCAlgorithm) -> None:
        super().__init__(brokerage_model, security_seeder)
        self.algorithm = algorithm

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, set the price model
        if security.Type == SecurityType.FutureOption: # Option type
            security.PriceModel = OptionPriceModels.CrankNicolsonFD()
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Filter Contracts

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 31 days

To adjust the universe of contracts, set a filter. The filter usually runs at every time step in your algorithm. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

To set a contract filter, in the `Initialize` method, pass a filter function to the `AddFutureOption` method. The following table describes the available filter techniques:

```
self.AddFutureOption(future.Symbol, lambda option_filter_universe: option_filter_universe.Strikes(-1, 1))
```

The following table describes the filter methods of the `OptionFilterUniverse` class:

Method	Description
`Strikes(minStrike: int, maxStrike: int)`	Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price
`CallsOnly()`	Selects call contracts
`PutsOnly()`	Selects put contracts
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weeklys contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```
self.AddFutureOption(future.Symbol, lambda option_filter_universe: option_filter_universe.Strikes(-1,
1).CallsOnly())
```

To perform thorough filtering on the `OptionFilterUniverse` , define an isolated filter method.

```
# In Initialize
self.AddFutureOption(future.Symbol, self.contract_selector)

def contract_selector(self, option_filter_universe: OptionFilterUniverse) -> OptionFilterUniverse:
    symbols = option_filter_universe.PutsOnly()
    strike = min([symbol.ID.StrikePrice for symbol in symbols])
    symbols = [symbol for symbol in symbols if symbol.ID.StrikePrice == strike]
    return option_filter_universe.Contracts(symbols)
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

Navigate Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security. They have the following properties:

To get the `OptionChain` , loop through the `OptionChains` property. After you get the `OptionChain` , you can sort and filter the Option contracts in the chain.

```
PY
def OnData(self, slice: Slice) -> None:
    for _, option_chain in slice.OptionChains.items():
        # Example: Find 5 put contracts that are closest to at-the-money (ATM) and have the farthest
expiration
        contracts = [x for x in option_chain if x.Right == OptionRight.Put]
        contracts = sorted(sorted(contracts, \
            key = lambda x: abs(option_chain.Underlying.Price - x.Strike)), \
            key = lambda x: x.Expiry, reverse=True)[:5]

        # Select the contract with the delta closest to -0.5
        contract = sorted(contracts, key=lambda x: abs(-0.5 - x.Greeks.Delta))[0]
```

You can also iterate through the `FuturesChains` first.

```
PY
def OnData(self, slice: Slice) -> None:
    for continuous_future_symbol, futures_chain in slice.FuturesChains.items():
        # Select a Future Contract and create its canonical FOP Symbol
        futures_contract = [contract for contract in futures_chain][0]
        canonical_fop_symbol = Symbol.CreateCanonicalOption(futures_contract.Symbol)
        fop_chain = slice.OptionChains.get(canonical_fop_symbol)
        if fop_chain:
            for contract in fop_chain:
                pass
```

Selection Frequency

By default, Future Option universes run at every time step to select their contracts. If you add the `OnlyApplyFilterAtMarketOpen` method to your contract filter, the universe selects contracts once a day at the first time step.

Live Trading Considerations

By default, LEAN adds contracts to the `OptionChain` that pass the filter criteria at every time step in your algorithm. If a contract has been in the universe for a duration that matches the `MinimumTimeInUniverse` setting and it no longer passes the filter criteria, LEAN removes it from the chain.

This default behavior can lead to a large number of open data subscriptions, which can be inconvenient in live trading since some data feed providers impose data subscription limits. To reduce the number of data subscriptions, change the value of the `MinimumTimeInUniverse` universe setting .

```
self.UniverseSettings.MinimumTimeInUniverse = timedelta(minutes=15)
self.UniverseSettings.MinimumTimeInUniverse = timedelta(0)
```

If you set the value to `timedelta(0)` , LEAN removes all of the contract subscriptions at every time step and your data provider stops sending the data. If your filter re-selects some of the same contracts when it runs next, LEAN requests new data subscriptions from your data provider. To avoid unsubscribing and resubscribing to the same contracts at every time step, use a `timedelta` greater than 0.

Universes

Index Options

Introduction

An Index Option universe lets you select a basket of Option contracts on an Index.

Create Universes

To add a universe of Index Option contracts, in the `Initialize` method, call the `AddIndexOption` method. This method returns an `Option` object, which contains the canonical `Symbol` . You can't trade with the canonical Option `Symbol` , but save a reference to it so you can easily access the Option contracts in the OptionChain that LEAN passes to the `OnData` method. The method to create the universe depends on if the Index Options you want require a target ticker.

Create Standard Universes

To create a universe of Index Options based on an index like VIX, SPX, or NDX, pass the index ticker to the `AddIndexOption` method.

```
option = self.AddIndexOption("VIX")
self.symbol = option.Symbol
```

The following table describes the `AddIndexOption` method arguments for standard universes:

Argument	Data Type	Description	Default Value
`ticker`	`str`	The underlying Index ticker. To view the supported indices, see Supported Assets .	
`resolution`	`Resolution/NoneType`	The resolution of the market data. To view the supported resolutions, see Resolutions .	`None`
`market`	`str`	The Index Option market.	`Market.USA`
`fillForward`	`bool`	If true, the current slice contains the last available data even if there is no data at the current time.	`True`

If you add an Option universe for an underlying Index that you don't have a subscription for, LEAN automatically subscribes to the underlying Index and sets its fill forward property to match that of the Index Option universe

Create Non-Standard Universes

To create a universe of non-standard Index Options like weekly VIX contracts, pass the index Symbol and target Option

ticker to the `AddIndexOption` method.

```
index_symbol = self.AddIndex("VIX").Symbol
option = self.AddIndexOption(index_symbol, "VIXW")
self.symbol = option.Symbol
```

The following table describes the `AddIndexOption` method arguments for non-standard universes:

Argument	Data Type	Description	Default Value
`underlying`	`Symbol`	The underlying Index `Symbol` . To view the supported indices, see Supported Assets .	
`targetOption`	`str`	The target Option ticker. To view the supported target Options, see Supported Assets .	
`resolution`	`Resolution/NoneType`	The resolution of the market data. To view the supported resolutions, see Resolutions . The Index resolution must be less than or equal to the Index Option resolution. For example, if you set the Index resolution to minute, then you must set the Index Option resolution to minute, hour, or daily.	`None`
`market`	`str`	The Index Option market.	`Market.USA`
`fillForward`	`bool`	If true, the current slice contains the last available data even if there is no data at the current time.	`True`

If you add an Option universe for an underlying Index that you don't have a subscription for, LEAN automatically subscribes to the underlying Index.

Configure Reality Models

To override the default pricing model of the Option, set a pricing model .

```
option.PriceModel = OptionPriceModels.CrankNicolsonFD()
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Filter Contracts

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 31 days

To adjust the universe of contracts, set a filter. The filter usually runs at every time step in your algorithm. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Option` object. The following table describes the available filter techniques:

Method	Description
`SetFilter(minStrike: int, maxStrike: int)`	Selects the contracts that have a strike price within a minimum and maximum strike level relative to the underlying price. For example, say the underlying price is 302*andthereare*strikesatevery5. If you set `minStrike` to -1 and `maxStrike` to 1, LEAN selects the contracts that have a strike of 300*or*305.
`SetFilter(minExpiry: timedelta, maxExpiry: timedelta)`	Selects the contracts that expire within the range you set.
`SetFilter(minStrike: int, maxStrike: int, minExpiry: timedelta, maxExpiry: timedelta)`	Selects the contracts that expire and have a strike within the range you set.
`SetFilter(universeFunc: Callable[[OptionFilterUniverse], OptionFilterUniverse])`	Selects the contracts that a function selects.

PY

```
# Select contracts that have a strike price within 1 strike level above and below the underlying price
option.SetFilter(minStrike=-1, maxStrike=1)

# Select contracts that expire within 30 days
option.SetFilter(minExpiry=timedelta(days=0), maxExpiry=timedelta(days=30))

# Select contracts that have a strike price within 1 strike level and expire within 30 days
option.SetFilter(minStrike=-1, maxStrike=1, minExpiry=timedelta(days=0), maxExpiry=timedelta(days=30))

# Select call contracts
option.SetFilter(lambda option_filter_universe: option_filter_universe.CallsOnly())
```

The following table describes the filter methods of the `OptionFilterUniverse` class:

Method	Description
`Strikes(minStrike: int, maxStrike: int)`	Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price
`CallsOnly()`	Selects call contracts
`PutsOnly()`	Selects put contracts
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weeklys contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```py
# Example 1: Select the front month call contracts
option.SetFilter(lambda option_filter_universe: option_filter_universe.CallsOnly().FrontMonth())

# Example 2: Select the contracts (including weeklys) that expire in the next 90 days
option.SetFilter(lambda option_filter_universe: option_filter_universe.IncludeWeeklys().Strikes(-20,
20).Expiration(0, 90))
```

To perform thorough filtering on the `OptionFilterUniverse` , define an isolated filter method.

```py
# In Initialize
option.SetFilter(self.contract_selector)

def contract_selector(self, option_filter_universe: OptionFilterUniverse) -> OptionFilterUniverse:
    symbols = option_filter_universe.PutsOnly()
    strike = min([symbol.ID.StrikePrice for symbol in symbols])
    symbols = [symbol for symbol in symbols if symbol.ID.StrikePrice == strike]
    return option_filter_universe.Contracts(symbols)
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

Navigate Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security. They have the following properties:

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` . After you get the `OptionChain` , you can sort and filter the Option contracts in the chain.

```py
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.symbol)
    if chain:
        # Example: Find 5 put contracts that are closest to at-the-money (ATM) and have the farthest
expiration
        contracts = [x for x in chain if x.Right == OptionRight.Put]
        contracts = sorted(sorted(contracts, \
            key = lambda x: abs(chain.Underlying.Price - x.Strike)), \
            key = lambda x: x.Expiry, reverse=True)[:5]

        # Select the contract with the delta closest to -0.5
        contract = sorted(contracts, key=lambda x: abs(-0.5 - x.Greeks.Delta))[0]
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```py
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.OptionChains.items():
        contracts = chain.Contracts
```

Selection Frequency

By default, Index Option universes run at every time step to select their contracts. If you add the `OnlyApplyFilterAtMarketOpen` method to your contract filter, the universe selects contracts once a day at the first time step.

Live Trading Considerations

By default, LEAN adds contracts to the `OptionChain` that pass the filter criteria at every time step in your algorithm. If a contract has been in the universe for a duration that matches the `MinimumTimeInUniverse` setting and it no longer passes the filter criteria, LEAN removes it from the chain.

This default behavior can lead to a large number of open data subscriptions, which can be inconvenient in live trading

since some data feed providers impose data subscription limits. To reduce the number of data subscriptions, change the value of the `MinimumTimeInUniverse` universe setting .

```
self.UniverseSettings.MinimumTimeInUniverse = timedelta(minutes=15)
self.UniverseSettings.MinimumTimeInUniverse = timedelta(0)
```

PY

If you set the value to `timedelta(0)` , LEAN removes all of the contract subscriptions at every time step and your data provider stops sending the data. If your filter re-selects some of the same contracts when it runs next, LEAN requests new data subscriptions from your data provider. To avoid unsubscribing and resubscribing to the same contracts at every time step, use a `timedelta` greater than 0.

Universes

Custom Universes

Introduction

A custom universe lets you select a basket of assets from a custom dataset.

Data Sources

You can gather your custom data from any of the following sources:

The data source should serve data in chronological order and each data point should have a unique timestamp. Each request has a 1 second overhead, so bundle samples together for fast execution.

Define Custom Universe Types

Custom universes should extend the `PythonData` class. Extensions of the `PythonData` class must implement a `GetSource` and `Reader` method.

The `GetSource` method in your custom data class instructs LEAN where to find the data. This method must return a `SubscriptionDataSource` object, which contains the data location and format (`SubscriptionTransportMedium`). You can even change source locations for backtesting and live modes. We support many different data sources.

The `Reader` method of your custom data class takes one line of data from the source location and parses it into one of your custom objects. You can add as many properties to your custom data objects as you need, but must set `Symbol` and `EndTime` properties. When there is no useable data in a line, the method should return `None` . LEAN repeatedly calls the `Reader` method until the date/time advances or it reaches the end of the file.

PY

```
# Example custom universe data; it is virtually identical to other custom data types.
class MyCustomUniverseDataClass(PythonData):

    def GetSource(self, config: SubscriptionDataConfig, date: datetime, isLiveMode: bool) ->
SubscriptionDataSource:
        return SubscriptionDataSource(@"your-remote-universe-data",
SubscriptionTransportMedium.RemoteFile)

    def Reader(self, config: SubscriptionDataConfig, line: str, date: datetime, isLiveMode: bool) ->
BaseData:
        items = line.split(",")

        # Generate required data, then return an instance of your class.
        data = MyCustomUniverseDataClass()
        data.EndTime = datetime.strptime(items[0], "%Y-%m-%d")
        # define Time as exactly 1 day earlier Time
        data.Time = data.EndTime - timedelta(1)
        data.Symbol = Symbol.Create(items[1], SecurityType.Crypto, Market.Bitfinex)
        data["CustomAttribute1"] = int(items[2])
        data["CustomAttribute2"] = float(items[3])
        return data
```

Your `Reader` method should return objects in chronological order. If an object has a timestamp that is the same or earlier than the timestamp of the previous object, LEAN ignores it.

If you need to create multiple objects in your `Reader` method from a single `line` , follow these steps:

1. In the `GetSource` method, pass `FileFormat.UnfoldingCollection` as the third argument to the `SubscriptionDataSource` constructor.
2. In the `Reader` method, order the objects by their timestamp and then return a `BaseDataCollection(endTime, config.Symbol, objects)` where `objects` is a list of your custom data objects.

```
class MyCustomUniverseDataClass(PythonData):

    def GetSource(self, config, date, isLive):
        return SubscriptionDataSource("your-data-source-url", SubscriptionTransportMedium.RemoteFile,
FileFormat.UnfoldingCollection)

    def Reader(self, config, line, date, isLive):
        json_response = json.loads(line)

        endTime = datetime.strptime(json_response[-1]["date"], '%Y-%m-%d') + timedelta(1)

        data = list()

        for json_datum in json_response:
            datum = MyCustomUniverseDataClass()
            datum.Symbol = Symbol.Create(json_datum["Ticker"], SecurityType.Equity, Market.USA)
            datum.Time = datetime.strptime(json_datum["date"], '%Y-%m-%d')
            datum.EndTime = datum.Time + timedelta(1)
            datum['CustomAttribute1'] = int(json_datum['Attr1'])
            datum.Value = float(json_datum['Attr1'])
            data.append(datum)

        return BaseDataCollection(endTime, config.Symbol, data)
```

Initialize Custom Universes

To add a custom universe to your algorithm, in the `Initialize` method, pass your universe type and a selector function to the `AddUniverse` method. The selector function receives a list of your custom objects and must return a list of `Symbol` objects. In the selector function definition, you can use any of the properties of your custom data type. The `Symbol` objects that you return from the selector function set the constituents of the universe.

```
# In Initialize
self.AddUniverse(MyCustomUniverseDataClass, "myCustomUniverse", Resolution.Daily, self.selector_function)

# Define the selector function
def selector_function(self, data: List[MyCustomUniverseDataClass]) -> List[Symbol]:
    sorted_data = sorted([ x for x in data if x["CustomAttribute1"] > 0 ],
                         key=lambda x: x["CustomAttribute2"],
                         reverse=True)
    return [x.Symbol for x in sorted_data[:5]]
```

Selection Frequency

Custom universes run on a schedule based on the `EndTime` of your custom data objects.

Examples

Demonstration Algorithms
DropboxUniverseSelectionAlgorithm.py Python

Universes

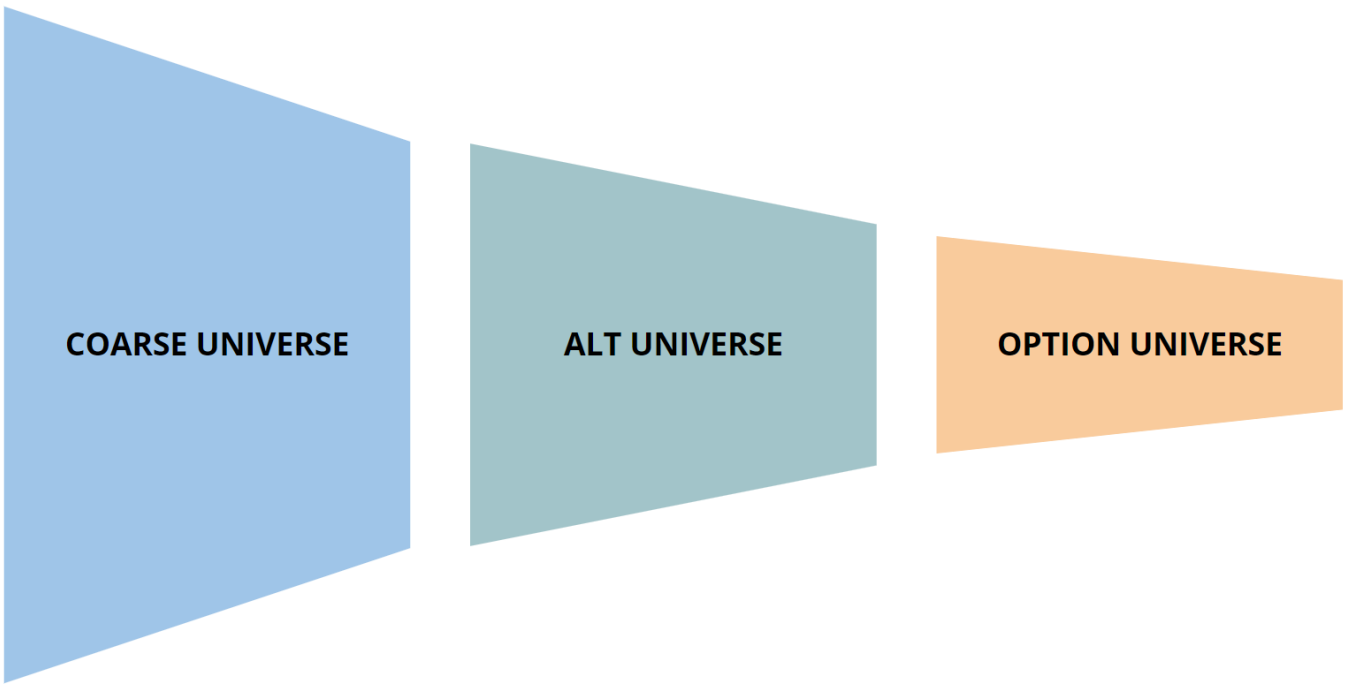
Chained Universes

Introduction

You can combine ("chain") universes together to fetch fundamental and alternative data on a specific subset of assets. Universes filter input data and return `Symbol` objects. The only requirement is that `Symbol` objects the filter returns are a subset of the input data. The source of the `Symbol` objects is unrestricted, so you can feed the output of one universe into another.

Filter Pattern

Universes filter a large set of Symbol objects by a coarse filter to quickly reduce the data processing requirement. This is often a first step before applying a second filter or requesting alternative data. For example, a strategy might only be interested in easily tradable liquid assets so quickly eliminates all stocks with less than $1M USD / day in trading volume.



The order of your filters can improve the speed of your research. By applying filters that narrow the universe the most, or are the lightest weight first, you can significantly reduce the amount of data your algorithm processes. Unless necessary, you can also not return any selections from earlier filters to further improve research speed, keeping only the universe data for later filters.

Universe Data Weights

To speed up your algorithm, request the lightest weight data first before chaining heavier filters or adding alternative data. The following table shows the size each dataset:

Name	Data Size / Weight
US Equities (Coarse)	Light (1 GB)
US Equities (Fine/Fundamental)	Heavy (5 TB)
US Equity Options	Huge (200 TB)
US Index Options	Medium (500 GB)
US Futures	Medium (500 GB)
US Futures Options	Medium (500 GB)
Crypto	Light (1 GB)
Alternative / General	Light (100 MB - 2 GB)
Alternative / Tiingo News	Medium (200 GB)

Universe Schedules

Universes typically run overnight and are available before market open. Universes are not currently "ordered", so universe chaining works best with slower universes. For example, use a slow-changing ETF Constituents Universe to set the Symbol list for alternative data.

Chaining Coarse and Alternative Data

The following example chains the QuiverQuantTwitterFollowersUniverse alternative universe to the coarse universe . It first selects the 100 most liquid US Equities and then filters them down based on their Twitter followers number and weekly change. The output of the alternative universe selection method is the output of the chained universe.

```
from AlgorithmImports import *

class ChainedUniverseAlgorithm(QCAlgorithm):

    coarse = []
    universe_coarse = None
    universe_twitter = None

    def Initialize(self):
        self.SetCash(100000)
        self.SetStartDate(2023, 1, 2)
        self.universe_coarse = self.AddUniverse(self.CoarseFilterFunction)
        self.universe_twitter = self.AddUniverse(QuiverQuantTwitterFollowersUniverse,
"QuiverQuantTwitterFollowersUniverse", Resolution.Daily, self.FollowerSelection)

    def CoarseFilterFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        sorted_by_dollar_volume = sorted(coarse, key=lambda x: x.DollarVolume, reverse=True)
        self.coarse = [c.Symbol for c in sorted_by_dollar_volume[:100]]
        return Universe.Unchanged

    def FollowerSelection(self, alt_coarse: List[QuiverQuantTwitterFollowersUniverse]) -> List[Symbol]:
        self.followers = [d.Symbol for d in alt_coarse if d.Followers > 200000 and d.WeekPercentChange >
0]
        return list(set(self.coarse) & set(self.followers))

    def OnSecuritiesChanged(self, changes):
        for added in changes.AddedSecurities:
            self.AddData(QuiverQuantTwitterFollowers, added.Symbol)

    def OnData(self, data):
        # Prices in the slice from the universe selection
        # Alternative data in slice from OnSecuritiesChanged Addition
        # for ticker,bar in data.Bars.items():
        #     pass
        for dataset_symbol, data_point in data.Get(QuiverQuantTwitterFollowers).items():
            self.Debug(f"{dataset_symbol} followers at {data.Time}: {data_point.Followers}")
```

Chaining ETF Universe and Alt Data

The following example chains a QuiverQuantTwitterFollowersUniverse alternative universe to the SPY ETF universe . It first selects all constituents of SPY and then filters them down with based on their Twitter followers number and weekly change. The output of the alternative universe selection method is the output of the chained universe.

```
PY
```

```python
from AlgorithmImports import *

class ChainedUniverseAlgorithm(QCAlgorithm):

    etf = []
    universe_etf = None
    universe_twitter = None

    def Initialize(self):
        self.SetCash(100000)
        self.SetStartDate(2023, 1, 2)

        self.universe_etf = self.AddUniverse(self.Universe.ETF("SPY", Market.USA, self.UniverseSettings,
self.ETFConstituentsFilter))
        # or symbol = Symbol.Create("SPY", SecurityType.Equity, Market.USA)
        # self.universe_etf = self.AddUniverseSelection(ETFConstituentsUniverseSelectionModel(
        #   symbol, self.UniverseSettings, self.ETFConstituentsFilter))
        self.universe_twitter = self.AddUniverse(QuiverQuantTwitterFollowersUniverse,
            "QuiverQuantTwitterFollowersUniverse", Resolution.Daily, self.FollowerSelection)

    def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
        self.etf = [c.Symbol for c in constituents]
        return Universe.Unchanged

    def FollowerSelection(self, alt_coarse: List[QuiverQuantTwitterFollowersUniverse]) -> List[Symbol]:
        self.followers = [d.Symbol for d in alt_coarse if d.Followers > 200000 and d.WeekPercentChange >
0]
        return list(set(self.etf) & set(self.followers))

    def OnSecuritiesChanged(self, changes):
        for added in changes.AddedSecurities:
            self.AddData(QuiverQuantTwitterFollowers, added.Symbol)

    def OnData(self, data):
        # Prices in the slice from the universe selection
        # Alternative data in slice from OnSecuritiesChanged Addition
        # for ticker,bar in data.Bars.items():
        #     pass
        for dataset_symbol, data_point in data.Get(QuiverQuantTwitterFollowers).items():
            self.Debug(f"{dataset_symbol} followers at {data.Time}: {data_point.Followers}")
```

Chaining to US Equity Options

The following example chains an `OptionFilterUniverse` to the QQQ ETF universe . It first selects the 30 largest-weighted constituents of QQQ and then selects their call Option contracts that expire within 60 days.

```
from AlgorithmImports import *

class ChainedUniverseAlgorithm(QCAlgorithm):

    universe_etf = None
    option_contracts = []

    def Initialize(self):
        self.SetCash(100000)
        self.SetStartDate(2023, 2, 2)

        self.UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw
        self.universe_etf = self.AddUniverse(self.Universe.ETF("QQQ", Market.USA, self.UniverseSettings,
self.ETFConstituentsFilter))

    def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
        sorted_by_weight = sorted(constituents, key=lambda x: x.Weight, reverse=True)
        return [c.Symbol for c in sorted_by_weight[:10]]

    def OnSecuritiesChanged(self, changes: SecurityChanges) -> None:
        for added in changes.AddedSecurities:
            if added.Type != SecurityType.Equity: continue

            contracts = self.OptionChainProvider.GetOptionContractList(added.Symbol, self.Time)
            filtered_contracts = [self.AddOptionContract(contract).Symbol for contract in contracts \
                                  if contract.ID.OptionRight == OptionRight.Call \
                                  and contract.ID.StrikePrice >= 100 \
                                  and contract.ID.Date <= self.Time + timedelta(10) \
                                  and contract.ID.OptionStyle == OptionStyle.American]
            self.option_contracts = self.option_contracts + filtered_contracts

        for removed in changes.RemovedSecurities:
            if removed.Symbol in self.option_contracts:
                self.option_contracts.remove(removed.Symbol)

    def OnData(self, data: Slice) -> None:
        for chain in data.OptionChains:
            symbol = chain.Key
            for contract in chain.Value:
                self.Debug(f"Found {contract.Symbol} option contract for {symbol}")
```

Chaining ETF and Fundamental Universe

The following example chains an ETF constituents universe with a fundamental universe to select ETF constituents with a low PE ratio.

PY

```
from AlgorithmImports import *

class ChainedUniverseAlgorithm(QCAlgorithm):

    universe_etf = None
    universe_fine = None

    def Initialize(self):
        self.SetCash(100000)
        self.SetStartDate(2023, 2, 2)

        self.universe_etf = self.Universe.ETF("QQQ", Market.USA, self.UniverseSettings,
self.ETFConstituentsFilter)
        self.universe_fine = self.AddUniverse(self.universe_etf, self.FineSelection)

    def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
        return [c.Symbol for c in constituents]

    def FineSelection(self, fine: List[FineFundamental]) -> List[Symbol]:
        sorted_by_pe_ratio = sorted(fine, key=lambda f: f.ValuationRatios.PERatio)
        return [f.Symbol for f in sorted_by_pe_ratio[:20]]

    def OnData(self, data):
        for symbol in data.Keys:
            self.Debug(f"{symbol} PE Ratio:
{self.Securities[symbol].Fundamentals.ValuationRatios.PERatio}")
```

Universes

Alternative Data Universes

Introduction

An alternative data universe lets you select a basket of assets based on an alternative dataset that's linked to securities. If you use an alternative data universe, you limit your universe to only the securities in the dataset, which avoids unnecessary subscriptions.

Supported Datasets

The following alternative datasets support universe selection:

- US Congress Trading
- Wikipedia Page Views
- WallStreetBets
- Corporate Buybacks
- Brain Sentiment Indicator
- Brain ML Stock Ranking
- Brain Language Metrics on Company Filings
- Twitter Followers
- CNBC Trading
- US Government Contracts
- Corporate Lobbying
- Insider Trading
- Crypto Market Cap

Datasets

Datasets

Overview

Introduction

It's common to use price and fundamental data points to inform trading decisions. Alternative datasets are datasets that aren't classified as market or fundamental data. Alternative data research began gaining popularity in 2007 after the financial crisis and its usage has continued to grow over the years. As usage has grown, so has the number of unique alternative datasets that are available for investors. The benefit of alternative datasets is that it’s easier to find alpha in these datasets because they receive less attention from investment researchers and these datasets contain information not found in market and fundamental datasets. Most traders research with market and fundamental datasets, so it's difficult to find alpha in them.

Methodology

The journey of raw data from its source to its integration with a trading platform can be complex. Alternative data is generally created through individuals, business processes, and sensors. Individuals create content on social media platforms, write reviews on e-commerce sites, and visit corporate Wikipedia pages. Businesses process transactions, file SEC reports, and issue company announcements. Sensors collect location data, gather satellite images, and monitor customer behavior.

Data providers can capture these forms of data for traders to inform trading decisions in ways that supplement traditional market and fundamental datasets. With over a million terabytes of data on the internet, there is virtually a limitless amount of information that can be informative for investors when analyzing a security’s performance in the marketplace.

Examples

A common example of alternative data in a trading system is using sentiment analysis on news sources to predict the future price movements of Equities. A second example is using announcements regarding corporate share buyback programs to trading Equities. A third example is using global supply and demand information for US Crude Products to inform Futures trades. To view all of the datasets we have available, see the left navigation menu.

Challenges

A challenge of working with datasets is that it can be difficult to find a reliable, high-quality, and inexpensive data feed to use in live algorithms. If a data feed has survivorship bias, insufficient history, or hasn’t been cleaned and processed properly, the data feed can cause poor trading performance. We thoroughly vet the datasets we add to the Dataset Market to ensure they are free of survivorship bias and have a reliable live feed. All of the available datasets have

already been cleaned and processed, so you can just focus on the strategy research and development.

Future Outlook

The Dataset Market is still in the early phases of development. Over time, we will add more markets and alternative datasets. Alternative datasets are an ever-expanding data source, so it's expected to continue to grow into the future. In today's digital world, more data is created each year than in the previous year. To take advantage of the trend, we have implemented a new onboarding process to integrate many new alternative data sources into the QuantConnect platform. With the Dataset Market, you can import a new alternative dataset into your trading algorithm with just a single line of code. The Dataset Market gives everyone the ability to access high-quality alternative datasets that were previously only accessible to major hedge funds.

Live Trading Considerations

We receive most daily alternative data at 7 AM Eastern Time (ET) and start processing it. In live trading, your algorithm receives this data within 30 minutes of 7 AM ET. In backtesting, this data has a timestamp of midnight, so your algorithm receives it about seven hours earlier than it does in live mode.

In backtests, your algorithm receives data at perfect timing. If you request minute resolution data, your algorithm receives the bars at the top of each minute. In live trading, bars have a slight delay, so you may receive them milliseconds after the top of each minute.

Note that there will be delay for data available to be backtested. Live feed data is available for backtesting after 24-48 hours normally. Consider using Paper live trading if close monitoring is needed.

Datasets

QuantConnect

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Binance Crypto Future Margin Rate Data

US ETF Constituents

US Equity Coarse Universe

US Equity Security Master

US Futures Security Master

QuantConnect

Binance Crypto Future Margin Rate Data

Introduction

The Binance Crypto Future Margin Rate Data by QuantConnect is for crypto-currency futures margin interest data points. The data covers 236 Cryptocurrency pairs, starts in August 2020, and is delivered on a daily update frequency. This dataset is created by downloading data using Binance API.

This dataset is an important companion to the Binance Crypto Future Price Data dataset because it contains information on margin interest data to model margin costs.

For more information about the Binance Crypto Future Margin Rate Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the Binance Crypto Future Margin Rate dataset:

```
PY
def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin)
    self.SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin)

    self.crypto_future_symbol = self.AddCryptoFuture("BTCBUSD", Resolution.Minute).Symbol
```

The Binance Crypto Future Margin Rate data is added to your algorithm along with the market data when you add a crypto future subscription.

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2020
Asset Coverage	236 Crypto Futures Pairs
Data Density	Regular
Resolution	Daily
Timezone	UTC

Supported Assets

The following table shows the available Crypto Future pairs:

Crypto Future Pairs					
1000LUNCBUSD	1000LUNCUSDT	1000SHIBBUSD	1000SHIBUSDT	1000XECUSDT	1INCHUSDT
AAVEUSD	AAVEUSDT	ADABUSD	ADAUSD	ADAUSDT	ALGOUSD
ALGOUSDT	ALICEUSDT	ALPHAUSDT	AMBBUSD	ANCBUSD	ANKRUSDT
ANTUSDT	APEBUSD	APEUSD	APEUSDT	API3USDT	APTBUSD
APTUSD	APTUSDT	ARPAUSDT	ARUSDT	ATAUSDT	ATOMUSD
ATOMUSDT	AUCTIONBUSD	AUDIOUSDT	AVAXBUSD	AVAXUSD	AVAXUSDT
AXSUSD	AXSUSDT	BAKEUSDT	BALUSDT	BANDUSDT	BATUSDT
BCHUSD	BCHUSDT	BELUSDT	BLUEBIRDUSDT	BLZUSDT	BNBBUSD
BNBUSD	BNBUSDT	BNXUSDT	BTCBUSD	BTCDOMUSDT	BTCUSD
BTCUSDT	BTSUSDT	C98USDT	CELOUSDT	CELRUSDT	CHRUSDT
CHZUSD	CHZUSDT	COMPUSDT	COTIUSDT	CRVUSDT	CTKUSDT
CTSIUSDT	CVCUSDT	CVXBUSD	CVXUSDT	DARUSDT	DASHUSDT
DEFIUSDT	DENTUSDT	DGBUSDT	DODOBUSD	DOGEBUSD	DOGEUSD
DOGEUSDT	DOTBUSD	DOTUSD	DOTUSDT	DUSKUSDT	DYDXUSDT
EGLDUSD	EGLDUSDT	ENJUSDT	ENSUSD	ENSUSDT	EOSUSD
EOSUSDT	ETCBUSD	ETCUSD	ETCUSDT	ETHBUSD	ETHUSD
ETHUSDT	FILBUSD	FILUSD	FILUSDT	FLMUSDT	FLOWUSDT
FOOTBALLUSDT	FTMBUSD	FTMUSD	FTMUSDT	FTTBUSD	FTTUSDT

Crypto Future Pairs					
GALABUSD	GALAUSD	GALAUSDT	GALBUSD	GALUSDT	GMTBUSD
GMTUSD	GMTUSDT	GRTUSDT	GTCUSDT	HBARUSDT	HNTUSDT
HOTUSDT	ICPBUSD	ICPUSDT	ICXUSD	ICXUSDT	IMXUSDT
INJUSDT	IOSTUSDT	IOTAUSDT	IOTXUSDT	JASMYUSDT	KAVAUSDT
KLAYUSDT	KNCUSD	KNCUSDT	KSMUSDT	LDOBUSD	LDOUSDT
LEVERBUSD	LINAUSDT	LINKBUSD	LINKUSD	LINKUSDT	LITUSDT
LPTUSDT	LRCUSDT	LTCBUSD	LTCUSD	LTCUSDT	LUNA2BUSD
LUNA2USDT	MANAUSD	MANAUSDT	MASKUSDT	MATICBUSD	MATICUSD
MATICUSDT	MKRUSDT	MTLUSDT	NEARBUSD	NEARUSD	NEARUSDT
NEOUSDT	NKNUSDT	OCEANUSDT	OGNUSDT	OMGUSDT	ONEUSDT
ONTUSDT	OPUSD	OPUSDT	PEOPLEUSDT	PHBBUSD	QNTUSDT
QTUMUSDT	RAYUSDT	REEFUSDT	RENUSDT	RLCUSDT	ROSEUSD
ROSEUSDT	RSRUSDT	RUNEUSD	RUNEUSDT	RVNUSDT	SANDBUSD
SANDUSD	SANDUSDT	SCUSDT	SFPUSDT	SKLUSDT	SNXUSDT
SOLBUSD	SOLUSD	SOLUSDT	SPELLUSDT	SRMUSDT	STGUSDT
STMXUSDT	STORJUSDT	SUSHIUSDT	SXPUSDT	THETAUSD	THETAUSDT
TLMBUSD	TLMUSDT	TOMOUSDT	TRBUSDT	TRXBUSD	TRXUSD
TRXUSDT	UNFIUSDT	UNIBUSD	UNIUSD	UNIUSDT	VETUSD
VETUSDT	WAVESBUSD	WAVESUSDT	WOOUSDT	XEMUSDT	XLMUSD
XLMUSDT	XMRUSD	XMRUSDT	XRPBUSD	XRPUSD	XRPUSDT
XTZUSD	XTZUSDT	YFIUSDT	ZECUSDT	ZENUSDT	ZILUSD
ZILUSDT	ZRXUSDT				

Data Point Attributes

The Binance Crypto Future Margin Rate dataset provides **MarginInterestRate** objects with the following attributes:

Requesting Data

To add Binance Crypto Future Margin Rate data to your algorithm, call the **AddCryptoFuture** method. Save a reference to the Crypto Future **Symbol** so you can access the data later in your algorithm.

```
class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)

        # Set Account Currency to Binance Stable Coin for USD
        self.SetAccountCurrency("BUSD")
        self.SetCash(100000)

        self.SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin)
        self.SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin)

        crypto_future = self.AddCryptoFuture("BTCBUSD", Resolution.Minute)
        # perpetual futures does not have a filter function
        self.symbol = crypto_future.Symbol
```

For more information about creating Crypto Future subscriptions, see Requesting Data .

Accessing Data

To get the current Binance Crypto Margin Rate data, index the **MarginInterestRates** property of the current **Slice** with the Crypto Future **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.MarginInterestRates:
        interest_rate = slice.MarginInterestRates[self.symbol].InterestRate
        self.Log(f"{self.symbol} close at {slice.Time}: {interest_rate}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, margin_interest_rate in slice.MarginInterestRates.items():
        interest_rate = margin_interest_rate.InterestRate
        self.Log(f"{symbol} close at {slice.Time}: {interest_rate}")
```

For more information about accessing Crypto Future data, see Handling Data .

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the **AddCrypto** method, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.symbol)
```

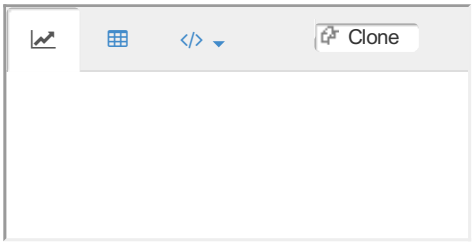
The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Example Applications

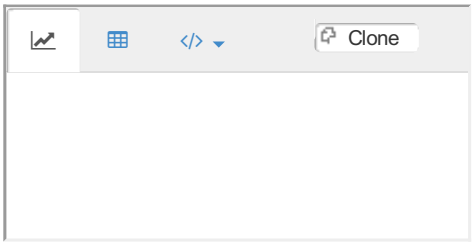
The Binance Crypto Future Margin Rate dataset enables correct margin cost so you can accurately design strategies for Cryptocurrencies with term structure. Examples include the following strategies:

- Horizontal/Diagonal arbitrage with the underlying cryptocurrencies
- Trade Contango/Backwardation predictions
- Hedge for illiquid cryptocurrencies

Python:



C#:



QuantConnect

US ETF Constituents

Introduction

The US ETF Constituents dataset by QuantConnect tracks the constituents and weighting of US Equities in 2,650 ETF listings. The data starts in January 2009 and is delivered on a daily basis. This dataset is created by tracking the host ETF websites and can be delayed by up to 1 week.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US ETF Constituents dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 160,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US ETF Constituents dataset:

```
PY
from QuantConnect.DataSource import *

# Use the following method for a Classic Algorithm
self.AddUniverse(self.Universe.ETF("SPY", Market.USA, self.UniverseSettings, self.ETFConstituentsFilter))

symbol = Symbol.Create("SPY", SecurityType.Equity, Market.USA)
# Use the following method for a Framework Algorithm
self.AddUniverseSelection(ETFConstituentsUniverseSelectionModel(symbol, self.UniverseSettings,
self.ETFConstituentsFilter))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2009
Asset Coverage	2,650 US ETF Listings
Data Density	Dense
Resolution	Daily
Timezone	New York

Supported ETFs

The following table shows the available ETFs:

ETF Tickers					
AADR	AAIT	AAWW	AAXJ	ABCS	ABEQ
ACCU	ACES	ACIM	ACIO	ACSG	ACSI
ACT	ACTV	ACWF	ACWI	ACWV	ACWX
ADFI	ADIV	ADME	ADRA	ADRD	ADRE
ADRU	AEMB	AESR	AFK	AFLG	AFMC
AFSM	AFTY	AGEM	AGG	AGGE	AGGH
AGGP	AGIH	AGII	AGLS	AGNG	AGOV
AGQ	AGRG	AGRH	AGT	AGZ	AHYB
AIA	AIEQ	AIIQ	AIQ	AIRR	ALFA
ALFI	ALTL	ALTS	ALTY	AMCA	AMER
AMID	AMLP	AMLX	AMOM	AMPS	AMZA
ANEW	ANGL	AOA	AOK	AOM	AOR
AOTG	APRZ	AQWA	ARCM	ARGT	ARKG
ARKK	ARKQ	ARKW	ARKX	ARVR	ASEA
ASET	ASHR	ASHX	AUGZ	AUSE	AUSF
AVDE	AVDV	AVEM	AVES	AVGE	AVIE
AVIV	AVLV	AVRE	AVSC	AVSD	AVSE
AVSU	AVUS	AVUV	AWAY	AXDI	AXEN

ETF Tickers					
AXFN	AXHE	AXID	AXIT	AXJL	AXJS
AXJV	AXMT	AXSL	AXTE	AXUT	BAB
BABS	BATT	BBAG	BBAX	BBC	BBCA
BBCB	BBEU	BBH	BBHY	BBIN	BBLU
BBMC	BBP	BBRC	BBRE	BBSA	BBSC
BBUS	BCHP	BCNA	BDAT	BDH	BDRY
BECO	BEDZ	BEMO	BERN	BETZ	BFIT
BFOR	BFTR	BHH	BIB	BIBL	BICK
BIDS	BIGD	BIK	BIKR	BITQ	BITS
BIV	BIZD	BJK	BKCH	BKCI	BKEM
BKES	BKF	BKGI	BKIE	BKIS	BKLC
BKMC	BKSE	BKUS	BLCN	BLDG	BLES
BLHY	BLKC	BLLD	BLOK	BLV	BMED
BND	BNDC	BNDW	BNDX	BNE	BNGE
BOB	BOSS	BOTZ	BOUT	BPAY	BRF
BRGL	BRNY	BRZU	BSCB	BSCC	BSCD
BSCE	BSCF	BSCG	BSCH	BSCI	BSCJ
BSCK	BSEA	BSJC	BSJD	BSJE	BSJF
BSJG	BSJH	BSJI	BSJJ	BSV	BTAH
BTAL	BTEC	BTEK	BTHM	BUFD	BUFF
BUFG	BUFQ	BUFR	BUFT	BUG	BUL
BUYZ	BUZ	BUZZ	BWX	BWZ	BYLD
BYOB	BYTE	CACG	CALF	CARZ	CATH
CBOE	CBON	CCON	CCOR	CCXE	CDC
CDEI	CDL	CDX	CEFA	CEFS	CEMB
CEY	CEZ	CFA	CFCV	CFGE	CFO

ETF Tickers					
CGDV	CGGO	CGGR	CGUS	CGW	CGXU
CHAU	CHB	CHEP	CHGX	CHIC	CHIH
CHII	CHIK	CHIL	CHIQ	CHIR	CHIS
CHIX	CHLD	CHNA	CIBR	CID	CIL
CIRC	CIU	CIZ	CLDL	CLIX	CLNR
CLOU	CLRG	CLSA	CLSC	CLSM	CLY
CLYH	CMBS	CMDT	CMF	CN	CNBS
CNCR	CNDA	CNDF	CNHX	CNRG	CNSF
CNTR	CNXT	COLX	COMB	COMG	COMT
CONG	COPX	COWZ	CPI	CQQQ	CRAK
CRBA	CRBI	CRBN	CRBQ	CRED	CRO
CROP	CRPT	CRUZ	CSA	CSB	CSD
CSF	CSJ	CSM	CSML	CTEC	CTEX
CTRU	CU	CUMB	CURE	CUT	CVIE
CVLC	CVMC	CVSE	CVY	CWEB	CWI
CWS	CXSE	CYA	CZA	DALI	DALT
DAPP	DAT	DAX	DBA	DBAP	DBAU
DBAW	DBB	DBBR	DBC	DBE	DBEF
DBEM	DBES	DBEU	DBEZ	DBGR	DBIF
DBIT	DBIZ	DBJP	DBKO	DBLV	DBMX
DBN	DBO	DBRE	DBSE	DBSP	DBU
DBUK	DDBI	DDEZ	DDIV	DDLS	DDM
DDWM	DEB	DECZ	DEEF	DEEP	DEF
DEFA	DEFN	DEHP	DEM	DEMG	DENT
DES	DESC	DEUS	DEW	DEWJ	DEZU
DFAC	DFAE	DFAI	DFAR	DFAS	DFAT

ETF Tickers					
DFAU	DFAX	DFE	DFEM	DFEN	DFEV
DFGR	DFHY	DFIC	DFIS	DFIV	DFJ
DFLV	DFND	DFNL	DFNV	DFSE	DFSI
DFSU	DFSV	DFUS	DFUV	DGRE	DGRO
DGRS	DGRW	DGS	DGT	DHDG	DHS
DIA	DIEM	DIG	DIHP	DIM	DINT
DISV	DIV	DIVA	DIVB	DIVD	DIVI
DIVO	DIVS	DIVZ	DJD	DJIA	DKA
DLN	DLS	DMAT	DMDV	DMRE	DMRI
DMRL	DMRM	DMRS	DMXF	DND	DNH
DNL	DOG	DOL	DON	DOO	DOZR
DPK	DPST	DQML	DRIV	DRLL	DRN
DRSK	DRW	DSC	DSCF	DSG	DSI
DSPC	DSTL	DSTX	DSUM	DSV	DTD
DTEC	DTH	DTN	DTOX	DTRE	DUDE
DUHP	DURA	DUSA	DUSL	DVAL	DVEM
DVLU	DVOL	DVP	DVY	DVYA	DVYE
DWAQ	DWAS	DWAT	DWAW	DWCR	DWEQ
DWFI	DWIN	DWLD	DWLV	DWM	DWMC
DWPP	DWSH	DWTR	DWUS	DWX	DXD
DXGE	DXJ	DXJC	DXJF	DXJH	DXJR
DXJS	DXJT	DXKW	DXPS	DXUS	DYHG
DYLD	DYLS	DYNF	DZK	EAFD	EAOA
EAOK	EAOM	EAOR	EASG	EASI	EATZ
EBIZ	EBLU	EBND	ECH	ECLN	ECNS
ECON	ECOW	ECOZ	EDBI	EDC	EDEN

ETF Tickers					
EDIV	EDOC	EDOG	EDOM	EDOW	EDUT
EEB	EEG	EEHB	EELV	EEM	EEMA
EEMD	EEME	EEML	EEMO	EEMS	EEMV
EEMX	EEN	EEO	EES	EEZ	EFA
EFAD	EFAV	EFAX	EFG	EFIV	EFN
EFNL	EFRA	EFV	EGIS	EGPT	EGRW
EGUS	EIDO	EINC	EIPX	EIRL	EIS
EKAR	EKG	EKH	ELG	ELR	ELV
EMAG	EMB	EMBB	EMBH	EMBU	EMCA
EMCD	EMCG	EMCH	EMCR	EMDD	EMDG
EMDI	EMDV	EMEM	EMER	EMEY	EMFM
EMFN	EMFQ	EMFT	EMG	EMGC	EMGD
EMGF	EMHD	EMHY	EMIF	EMLC	EMLP
EMM	EMMT	EMPW	EMQQ	EMRE	EMSG
EMSO	EMT	EMTL	EMV	EMXC	EMXF
EMZA	ENFR	ENGN	ENOR	ENRG	ENTR
ENY	ENZL	EOPS	EPHE	EPI	EPOL
EPP	EPRE	EPRF	EPRO	EPS	EPU
EQAL	EQIN	EQL	EQLT	EQRR	EQUL
EQWL	EQWM	EQWS	ERET	ERGF	ERM
ERSX	ERUS	ERX	ESG	ESGA	ESGD
ESGE	ESGF	ESGG	ESGL	ESGN	ESGS
ESGU	ESGV	ESGW	ESGY	ESIX	ESML
ESMV	ESNG	ESPO	ESR	ETHO	ETPA
EUDG	EUDV	EUFL	EUFN	EUMF	EUMV
EURL	EURZ	EUSA	EUSC	EUXL	EVAL

ETF Tickers					
EVEN	EVX	EWA	EWAC	EWC	EWCO
EWD	EWEB	EWEF	EWEM	EWG	EWGS
EWH	EWHS	EWI	EWJ	EWK	EWL
EWM	EWMC	EWMD	EWN	EWO	EWP
EWQ	EWRE	EWRI	EWRM	EWRS	EWS
EWSC	EWSM	EWSS	EWT	EWU	EWUS
EWW	EWX	EWY	EWZ	EWZS	EXB
EXI	EXT	EYLD	EZA	EZJ	EZM
EZU	EZY	FAA	FAB	FAD	FAIL
FAN	FAS	FATT	FAUS	FBCG	FBCV
FBM	FBND	FBT	FBZ	FCA	FCAN
FCD	FCEF	FCG	FCHI	FCL	FCLD
FCOM	FCOR	FCPI	FCQ	FCTR	FCV
FCVT	FDD	FDEM	FDEV	FDG	FDHT
FDIG	FDIS	FDIV	FDL	FDLO	FDM
FDMO	FDN	FDNI	FDRR	FDRV	FDT
FDTS	FDV	FDVV	FDWM	FEBZ	FEDM
FEDX	FEEM	FEFN	FEG	FEHY	FEIG
FEM	FEMB	FEMS	FENY	FEP	FEUS
FEUZ	FEVR	FEX	FFHG	FFL	FFR
FFSG	FFTG	FFTI	FFTY	FGD	FGEM
FGM	FGRO	FHC	FHK	FHLC	FIDI
FIDU	FIL	FILL	FINU	FINX	FIO
FISR	FITE	FIVA	FIVG	FIW	FJP
FKL	FKO	FKU	FLAG	FLAU	FLAX
FLBR	FLCA	FLCH	FLCO	FLEE	FLEH

ETF Tickers					
FLFR	FLG	FLGB	FLGR	FLHK	FLIN
FLIO	FLIY	FLJH	FLJP	FLKR	FLLA
FLLV	FLM	FLMB	FLMI	FLMX	FLN
FLOT	FLQD	FLQE	FLQG	FLQH	FLQL
FLQM	FLQS	FLRG	FLRT	FLRU	FLSA
FLSP	FLSW	FLTB	FLTR	FLTW	FLV
FLYT	FLZA	FM	FMAG	FMAT	FMB
FMET	FMF	FMIL	FMK	FMM	FMU
FMV	FNCF	FNCL	FNDA	FNDB	FNDC
FNDE	FNDF	FNDX	FNG	FNGG	FNI
FNIO	FNK	FNTC	FNX	FNY	FOC
FOMO	FONE	FOS	FOVL	FPA	FPE
FPFD	FPRO	FPX	FPXE	FPXI	FQAL
FRAK	FRDM	FREL	FRI	FRL	FRN
FRNW	FSBD	FSMD	FSMO	FSST	FSTA
FSYD	FSZ	FTA	FTAG	FTC	FTCS
FTEC	FTGC	FTGS	FTHI	FTLB	FTLS
FTQ	FTQI	FTRI	FTSL	FTSM	FTVA
FTW	FTXD	FTXG	FTXH	FTXL	FTXN
FTXO	FTXR	FTY	FUI	FUTY	FV
FVAL	FVC	FVD	FVI	FVL	FWDB
FWDD	FWDI	FXD	FXEU	FXG	FXH
FXI	FXL	FXN	FXO	FXR	FXU
FXZ	FYC	FYLG	FYT	FYX	FZB
GAA	GAF	GAL	GAMR	GARD	GASL
GASX	GBDV	GBF	GBGR	GBIL	GBLD

ETF Tickers					
GBLO	GBUY	GCC	GCOR	GCOW	GDAT
GDIV	GDJJ	GDMA	GDNA	GDOC	GDVD
GDX	GDXJ	GDXX	GEM	GEMD	GENY
GERJ	GERM	GEUR	GEX	GFGF	GFIN
GFOF	GGEM	GHII	GHS	GHYB	GHYG
GIGB	GIGE	GII	GINN	GIVE	GIY
GK	GLCN	GLDE	GLIF	GLIN	GLOF
GLOV	GLRY	GMAN	GMET	GMF	GMFL
GMFS	GML	GMM	GMOM	GMTB	GNAT
GNMA	GNOM	GNR	GNRX	GOAT	GOAU
GOEX	GOP	GOVT	GPAL	GQRE	GREI
GREK	GRES	GRI	GRID	GRMY	GRNB
GRNR	GRPC	GRV	GSAX	GSD	GSEE
GSEU	GSEW	GSFP	GSG	GSGO	GSID
GSIE	GSIG	GSJY	GSLC	GSMA	GSPY
GSRA	GSSC	GSST	GSUS	GSY	GTAA
GTEK	GTIP	GTO	GULF	GUNR	GUR
GURU	GUSA	GUSH	GVAL	GVI	GVIP
GVT	GWL	GWX	GXC	GXF	GXG
GXTG	GYEN	GYLD	HACK	HACV	HACW
HAHA	HAIL	HAO	HAP	HAPI	HAPY
HART	HAUD	HAUZ	HAWX	HBTA	HBU
HCOM	HCRF	HDAW	HDEE	HDEF	HDEZ
HDG	HDGE	HDGI	HDIV	HDLS	HDMV
HDRO	HDUS	HDV	HDWM	HDWX	HECO
HEDJ	HEEM	HEET	HEFA	HEFV	HEGE

ETF Tickers					
HEGJ	HELX	HEMV	HEQT	HERD	HERO
HEUS	HEUV	HEWC	HEWG	HEWI	HEWJ
HEWL	HEWP	HEWU	HEWW	HEWY	HEZU
HFEZ	HFGO	HFND	HFXE	HFXI	HFXJ
HGEM	HGEU	HGI	HGSD	HHH	HIBL
HIDE	HILO	HIPR	HIPS	HISF	HJEN
HJPX	HKK	HLAL	HLGE	HMTM	HNDL
HOLD	HOM	HOMZ	HOTL	HSCZ	HSMV
HSPX	HTEC	HTUS	HUSE	HUSV	HVAL
HVOL	HVPW	HYD	HYDD	HYDR	HYEM
HYG	HYGH	HYGI	HYGV	HYHG	HYIH
HYLD	HYLG	HYLS	HYLV	HYXU	IAH
IAI	IAK	IAT	IAU	IBB	IBBJ
IBBQ	IBCB	IBCC	IBCD	IBCE	IBD
IBDA	IBDB	IBDC	IBDD	IBDF	IBDH
IBDJ	IBDK	IBDL	IBDM	IBDN	IBDO
IBDP	IBDQ	IBDT	IBHA	IBHB	IBHC
IBHD	IBHE	IBLC	IBLN	IBMD	IBME
IBMF	IBMG	IBMH	IBMI	IBND	IBRN
IBUY	ICAN	ICAP	ICF	ICLN	ICOL
ICOW	ICSH	ICVT	IDAT	IDEV	IDHB
IDHD	IDHQ	IDLB	IDLV	IDMO	IDNA
IDOG	IDRV	IDU	IDV	IDVO	IDX
IECS	IEDI	IEF	IEFA	IEFN	IEHS
IEI	IEIH	IEIL	IEIS	IELG	IEME
IEMG	IEO	IESM	IETC	IEUR	IEUS

ETF Tickers					
IEV	IEZ	IFAS	IFEU	IFGL	IFLY
IFNA	IFRA	IFSM	IFV	IGBH	IGE
IGEM	IGF	IGHG	IGIB	IGM	IGN
IGOV	IGRO	IGSB	IGU	IGV	IGW
IHAK	IHDG	IHE	IHF	IHI	IHY
IIH	IJH	IJJ	IJK	IJNK	IJR
IJS	IJT	ILDR	ILF	ILTB	IMFL
IMOM	IMTM	INCO	INDA	INDF	INDL
INDS	INDY	INFL	INFR	INKM	INNO
INTF	INXX	IOIL	IOO	IPAC	IPAY
IPD	IPF	IPFF	IPK	IPKW	IPN
IPO	IPOS	IPS	IPU	IPW	IQDE
IQDF	IQDG	IQDY	IQIN	IQLT	IQM
IQSI	IQSU	IRBO	IRO	IRV	IRY
ISCF	ISDS	ISDX	ISEM	ISHG	ISHP
ISI	ISMD	ISRA	IST	ISTB	ISVL
ISZE	ITA	ITAN	ITB	ITEQ	ITF
ITIP	ITM	ITOT	IUS	IUSB	IUSG
IUSS	IUSV	IVAL	IVDG	IVE	IVEG
IVES	IVLC	IVLU	IVOG	IVOL	IVOO
IVOV	IVRA	IVSG	IVV	IVW	IWB
IWC	IWD	IWF	IWFH	IWIN	IWL
IWM	IWN	IWO	IWP	IWR	IWS
IWTR	IWV	IWW	IWX	IWY	IWZ
IXC	IXG	IXJ	IXN	IXP	IXUS
IYC	IYE	IYF	IYG	IYH	IYJ

ETF Tickers					
IYK	IYLD	IYM	IYR	IYT	IYW
IYY	IYZ	IZRL	JANZ	JAVA	JBBB
JCO	JCPB	JCPI	JCTR	JDG	JDIV
JEMA	JEPI	JETS	JGLD	JGRO	JHCB
JHCS	JHDV	JHEM	JHID	JHMA	JHMC
JHMD	JHME	JHMF	JHMH	JHMI	JHML
JHMM	JHMS	JHMT	JHMU	JHPI	JHSC
JIDA	JIG	JIRE	JKD	JKE	JKF
JKG	JKH	JKI	JKJ	JKK	JKL
JMBS	JMEE	JMIN	JMOM	JMST	JMUB
JNUG	JOET	JPED	JPEM	JPEU	JPGB
JPGE	JPHF	JPHY	JPIE	JPIN	JPMB
JPME	JPMV	JPNL	JPRE	JPSE	JPST
JPUS	JQUA	JRE	JRNY	JSCP	JSMD
JSML	JUSA	JUST	JVAL	JXI	JZRO
KALL	KARS	KBA	KBE	KBUY	KBWB
KBWC	KBWD	KBWI	KBWP	KBWR	KBWX
KBWY	KCE	KCNY	KDFI	KDIV	KEJI
KEMP	KEMQ	KEMX	KESG	KFVG	KFYP
KGHG	KGRN	KGRO	KIE	KLCD	KLD
KLDW	KLEM	KLIP	KLNE	KME	KMED
KMLM	KNG	KNGS	KNOW	KOCG	KOIN
KOKU	KOL	KOMP	KORP	KORU	KRBN
KRE	KRMA	KROO	KROP	KRU	KSCD
KURE	KVLE	KWEB	KWT	KXI	LABU
LALT	LATM	LBJ	LBTA	LCTD	LCTU

ETF Tickers					
LDEM	LDRI	LDRS	LDSF	LEAD	LEGR
LEMB	LEND	LFEQ	LGEM	LGLV	LGOV
LIT	LIV	LKOR	LLDM	LLEM	LLSC
LLSP	LMBS	LNGR	LOPX	LOUP	LOWC
LQD	LQDH	LRGE	LRGF	LRNZ	LSAF
LSAT	LSLT	LTL	LUXE	LVHB	LVHD
LVHE	LVHI	LVIN	LVL	LVOL	LVUS
LWPE	LYFE	MAAX	MAGA	MAKX	MAMB
MARB	MARZ	MATF	MATH	MATL	MAUI
MBB	MBCC	MBOX	MBSD	MCEF	MCHI
MCRO	MCSE	MDCP	MDD	MDEV	MDIV
MDLL	MDY	MDYG	MDYV	MEAR	MEME
MES	META	METV	MEXX	MFDX	MFEM
MFMS	MFUL	MFUS	MGC	MGK	MGV
MID	MIDE	MIDF	MIDU	MIDZ	MILN
MINC	MISL	MJ	MJUS	MKH	MLN
MLPA	MLPX	MMIN	MMIT	MMLG	MMSC
MMTM	MNA	MNM	MOAT	MOHR	MOM
MONY	MOO	MOOD	MOON	MORT	MOTE
MOTG	MOTI	MOTO	MPRO	MRAD	MRGR
MSGR	MSOS	MTK	MTUM	MUB	MULT
MUSI	MVIN	MVP	MVPS	MVV	MXDU
MXI	MZG	MZN	MZO	NACP	NAIL
NANR	NASH	NASI	NDIV	NDJI	NDVG
NEAR	NEED	NERD	NETL	NFLT	NFO
NFRA	NIFE	NIWM	NLR	NOBL	NOMO

ETF Tickers					
NORW	NSCS	NSPI	NSPY	NTKI	NTSE
NTSI	NUAG	NUBD	NUCL	NUDM	NUDV
NUEM	NUGO	NUGT	NUHY	NULC	NULG
NULV	NUMG	NUMV	NURE	NUSA	NUSC
NUSI	NVQ	NWLG	NXTE	NXTG	NY
NYC	NYCC	NYF	NZAC	OASI	OBOR
OCEN	OCIO	OEF	OEUR	OEW	OGEM
OGIG	OIH	OLD	OMFL	OMFS	OMOM
OND	ONEF	ONEK	ONEO	ONEQ	ONEV
ONEY	ONG	ONLN	ONOF	ONTL	OOTO
OPD	OQAL	ORG	OSCV	OSIZ	OTP
OTR	OUSA	OUSM	OVB	OVF	OVL
OVLC	OVLH	OVLU	OVM	OVOL	OVS
OVT	OWNS	OYLD	PABU	PACA	PAF
PAGG	PALC	PAMC	PAO	PAVE	PAWZ
PBD	PBDM	PBE	PBEE	PBJ	PBP
PBS	PBSM	PBTQ	PBUS	PBW	PCA
PCEF	PDBC	PDEV	PDN	PDP	PEGA
PEJ	PEK	PEX	PEXL	PEY	PEZ
PFA	PFF	PFFA	PFFD	PFFR	PFI
PFIG	PFIX	PFM	PFUT	PFXF	PGAL
PGF	PGHY	PGJ	PGRO	PGX	PHB
PHDG	PHO	PHYL	PIC	PICB	PICK
PID	PIE	PILL	PIN	PINK	PIO
PIQ	PIV	PIZ	PJB	PJF	PJG
PJM	PJP	PKB	PKN	PKOL	PKW

ETF Tickers

PLAT	PLCY	PLDR	PLND	PLRG	PLTL
PLTM	PLW	PMA	PMNA	PMOM	PMPT
PMR	PNQI	PNXQ	POTX	PPA	PPH
PPI	PPLC	PPSC	PQBW	PQDI	PQIN
PQLC	PQSC	PQSG	PQSV	PQY	PQZ
PRAY	PREF	PRF	PRFZ	PRME	PRN
PRNT	PSAU	PSC	PSCC	PSCD	PSCE
PSCF	PSCH	PSCI	PSCM	PSCT	PSCU
PSDN	PSET	PSFF	PSI	PSIL	PSJ
PSK	PSL	PSMB	PSMC	PSMG	PSMM
PSP	PSQ	PSR	PSTL	PSY	PTE
PTEU	PTF	PTH	PTIN	PTJ	PTLC
PTMC	PTNQ	PTO	PTRP	PUI	PUTW
PUW	PVAL	PVI	PWB	PWC	PWJ
PWND	PWO	PWP	PWT	PWV	PWY
PWZ	PXE	PXF	PXH	PXI	PXJ
PXLC	PXLG	PXLV	PXMC	PXMG	PXMV
PXN	PXQ	PXR	PXSC	PXSG	PXSV
PXUS	PY	PYH	PYZ	PZA	PZD
PZI	PZJ	PZT	QABA	QAI	QARP
QAT	QCAN	QCLN	QCLR	QDEF	QDEM
QDF	QDIV	QDPL	QDXU	QDYN	QED
QEFA	QEH	QEM	QEMM	QGBR	QGEM
QGRO	QGTA	QID	QINC	QINT	QLC

ETF Tickers					
QLD	QLS	QLT	QLTA	QLTB	QLTC
QLV	QLVD	QLVE	QMJ	QMN	QMOM
QPT	QPX	QQC	QQD	QQEW	QQJG
QQMG	QQQ	QQQA	QQQE	QQQJ	QQQM
QQQN	QQQQ	QQXT	QRFT	QRMI	QSY
QTEC	QTR	QTUM	QUAL	QUS	QVAL
QVM	QVML	QVMM	QVMS	QWLD	QXGG
QXMI	QXRR	QXTR	QXUS	QYLD	QYLG
RAAX	RAFE	RALS	RAVI	RAYS	RBIN
RBL	RBLD	RBUS	RCD	RDIV	RDMX
RDOG	RDVI	RDVY	RECS	REDV	REEM
REET	REFA	REGL	REIT	REM	REMG
REMX	RESE	RESI	RESP	RETL	REVS
REZ	RFAP	RFDA	RFDI	RFEM	RFEU
RFF	RFFC	RFG	RFUN	RFV	RGI
RGLB	RGRO	RHS	RIDV	RIET	RIGS
RIGZ	RING	RISN	RKH	RLY	RNDM
RNDV	RNEM	RNLC	RNMC	RNRG	RNSC
ROAM	ROB	ROBO	ROBT	ROCI	RODE
RODM	ROGS	ROI	ROKT	ROM	ROMO
ROOF	RORE	RORO	ROSC	ROUS	RPG
RPQ	RPV	RPX	RRF	RRGR	RSCO
RSP	RSPE	RSPY	RSU	RSUN	RSX
RSXJ	RTH	RTL	RTM	RTR	RTYD
RULE	RUSL	RUSS	RVRS	RWCD	RWDC

ETF Tickers					
RWDE	RWED	RWG	RWGV	RWIU	RWJ
RWK	RWL	RWLS	RWM	RWO	RWR
RWSL	RWUI	RWV	RWVG	RWW	RWX
RXI	RXL	RYE	RYF	RYH	RYJ
RYLD	RYLG	RYT	RYU	RZG	RZV
SAA	SAEF	SAGG	SATO	SAVN	SBEU
SBIO	SBUS	SCAP	SCHA	SCHB	SCHC
SCHD	SCHE	SCHF	SCHG	SCHH	SCHI
SCHJ	SCHK	SCHM	SCHO	SCHP	SCHQ
SCHR	SCHV	SCHX	SCHY	SCHZ	SCIF
SCIN	SCJ	SCLP	SCOG	SCTR	SCZ
SDAG	SDCI	SDEM	SDG	SDGA	SDIV
SDOG	SDOW	SDS	SDSI	SDVY	SDY
SEA	SECT	SEMI	SENT	SEPZ	SFY
SFYF	SFYX	SGDJ	SGDM	SGGG	SGQI
SH	SHBT	SHE	SHLD	SHMO	SHNY
SHOC	SHV	SHVY	SHY	SHYD	SHYG
SIL	SILJ	SIMS	SIPE	SIXA	SIXH
SIXL	SIXS	SIZ	SIZE	SKOR	SKYY
SLBT	SLDR	SLIM	SLOW	SLQD	SLT
SLVP	SLVY	SLX	SLY	SLYG	SLYV
SMB	SMCP	SMD	SMDV	SMDY	SMH
SMIG	SMIN	SMLE	SMLF	SMLL	SMLV
SMMD	SMMV	SMOG	SNPE	SNSR	SNUG
SOCL	SOLR	SOVB	SOXL	SOXQ	SOXX

ETF Tickers					
SPAK	SPBC	SPCX	SPD	SPDV	SPDW
SPEM	SPEU	SPGM	SPGP	SPHB	SPHD
SPHQ	SPLG	SPLV	SPMD	SPMO	SPMV
SPQQ	SPRE	SPSB	SPSM	SPTM	SPUC
SPUN	SPUS	SPUU	SPVM	SPVU	SPXE
SPXH	SPXL	SPXN	SPXS	SPXT	SPXU
SPXV	SPY	SPYB	SPYC	SPYD	SPYG
SPYV	SPYX	SQEW	SQLV	SQQQ	SRET
SRVR	SSAM	SSFI	SSLY	SSO	SSPX
SSPY	SSUS	SSXU	STH	STIP	STLC
STLG	STLV	STMB	STOT	STRV	STSB
STXD	STXG	STXK	STXV	STXX	SUB
SUBZ	SULR	SUNY	SUPL	SURE	SUSA
SUSC	SUSL	SVAL	SVOL	SWAN	SWAR
SWH	SWIN	SXQG	SXUS	SYE	SYG
SYLD	SYTL	SYUS	SYV	SZNE	TAIL
TAN	TAO	TAWK	TAXF	TAXR	TBF
TBLU	TBT	TCHF	TCHI	TCHP	TCTL
TDD	TDH	TDIV	TDN	TDSA	TDSB
TDSC	TDSD	TDSE	TDTF	TDTT	TDV
TDVG	TDX	TECB	TECL	TEMP	TENG
TEQI	TERM	TETF	TFIV	TFLO	TFLT
TGEM	TGN	TGR	TGRW	THCX	THD
THHY	THNQ	THRK	TILT	TINT	TINY
TIP	TIPX	TLDH	TLEH	TLH	TLT

ETF Tickers					
TLTD	TLTE	TMDV	TMF	TMFC	TMFE
TMFG	TMFM	TMFS	TMFX	TMW	TNA
TOK	TOKE	TOLZ	TONS	TOTL	TPAY
TPHD	TPHE	TPIF	TPLC	TPLE	TPOR
TPSC	TPYP	TQQQ	TRFM	TRND	TRPL
TRSK	TRTY	TSPA	TTAC	TTAI	TTFS
TTH	TUR	TUSA	TUTI	TUTT	TWEB
TWM	TWOK	TWON	TYD	TYLG	TYNE
TYNS	TZD	TZE	TZG	TZI	TZL
TZO	TZV	TZW	TZY	UAE	UAV
UBD	UBIO	UBOT	UBT	UCC	UCOM
UCYB	UDBI	UDIV	UDOW	UEVM	UGE
UGEM	UIVM	UJB	UK	UKF	UKK
UKW	ULQ	ULST	ULTR	ULVM	UMDD
UMI	UOP	UPRO	UPW	UPWD	URA
URE	URTH	URTY	USAI	USD	USEQ
USHG	USIG	USLB	USMC	USMF	USMV
USPX	USRT	USSD	USSG	UST	USVM
USWD	USXF	UTES	UTH	UTLF	UTLT
UTRN	UTSL	UUP	UVDV	UVG	UVT
UVU	UWC	UWM	UXI	UYG	UYM
VALQ	VALX	VAMO	VAW	VB	VBK
VBR	VCAR	VCIT	VCLN	VCLO	VCLT
VCR	VCSH	VDC	VDE	VEA	VEGA
VEGI	VERS	VESH	VETS	VEU	VFH

ETF Tickers					
VFIN	VFLQ	VFMF	VFMO	VFMV	VFQY
VFVA	VGEM	VGFO	VGIT	VGK	VGSH
VGT	VHT	VICE	VIG	VIGI	VIOG
VIOO	VIOV	VIRS	VIS	VIXH	VLLV
VLML	VLSM	VLU	VLUE	VMOT	VNAM
VNLA	VNM	VNQ	VNQI	VO	VOE
VONE	VONG	VONV	VOO	VOOG	VOOV
VOT	VOX	VPC	VPL	VPN	VPOP
VPU	VR	VRAI	VRP	VSDA	VSGX
VSL	VSMV	VSPY	VSS	VT	VTC
VTHR	VTI	VTRN	VTV	VTWG	VTWO
VTWV	VUG	VV	VWID	VWO	VXF
VXUS	VYM	VYMI	WANT	WBAL	WBIA
WBIB	WBIC	WBID	WBIE	WBIF	WBIG
WBIH	WBII	WBIL	WBIN	WBIR	WBIT
WBIY	WCAT	WCBR	WCHN	WCLD	WDIV
WDNA	WDRW	WDTI	WEAR	WEBL	WEXP
WFH	WFVK	WGRO	WINN	WIP	WIZ
WKLY	WLDR	WLTH	WMCR	WMH	WNDY
WOMN	WOOD	WPS	WREI	WSKY	WTAI
WTRE	WTRX	WTV	WUGI	WUSA	WWJD
WWOW	WXSP	XAR	XB	XBB	XBI
XBUY	XCCC	XCEM	XCLR	XCOM	XDAT
XDNA	XES	XFIV	XGC	XHB	XHE
XHLF	XHMO	XHS	XHYC	XHYD	XHYE
XHYF	XHYH	XHYI	XHYT	XITK	XJH

ETF Tickers					
XJR	XKFS	XKII	XKST	XLB	XLBS
XLBT	XLC	XLE	XLES	XLF	XLFS
XLG	XLI	XLIS	XLK	XLKS	XLP
XLPS	XLRE	XLSR	XLU	XLUS	XLV
XLVO	XLVS	XLY	XLYS	XME	XMHQ
XMLV	XMMO	XMPT	XMVM	XMX	XNTK
XONE	XOP	XOUT	XPH	XPND	XRLV
XRMI	XRO	XRT	XSD	XSHD	XSHQ
XSLV	XSMO	XSOE	XSVM	XSVN	XSW
XT	XTEN	XTH	XTL	XTN	XTR
XTRE	XTWO	XTWY	XUSA	XVOL	XVV
XWEB	XYLD	XYLG	YAO	YDIV	YESR
YINN	YLCO	YLD	YLDE	YMLI	YMLP
YOLO	YPRO	YPS	YUMY	YYY	ZCAN
ZDEU	ZGBR	ZGEN	ZHOK	ZIG	ZLRG
ZMLP	ZSML				

Data Point Attributes

The ETF Constituents dataset provides **ETFConstituentData** objects, which have the following attributes:

Requesting Data

To add US ETF Constituents data to your algorithm, call the **AddUniverse** and **Universe.ETF** methods.. To select which constituents occupy the universe, provide the ETF **Symbol** and a selection function.

```
class ETFConstituentsDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2018, 1, 1)
        self.SetEndDate(2020, 8, 25)
        self.SetCash(100000)

        self.AddUniverse(self.Universe.ETF("SPY", self.UniverseSettings, self.ETFConstituentsFilter))
```

Accessing Data

To access the US ETF Constituent data, use the **ETFConstituentData** objects in your selection function. The data is available in daily resolution. The **Symbol** objects you return from your selection function defines the universe constituents.

```
PY
def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
    for c in constituents:
        self.Debug(f'{c.EndTime} :: {c.LastUpdate} :: {c.Weight} :: {c.SharesHeld} :: {c.MarketValue}')
    return [x.Symbol for x in constituents]
```

Historical Data

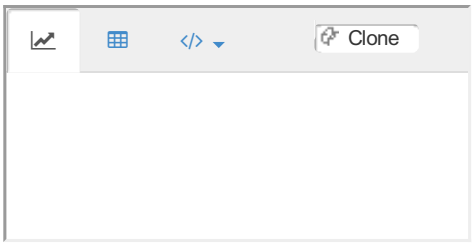
You can't request historical **ETFConstituentData** objects, but you can you can use the **Symbol** member of each object to request historical market data of the securities that the **ETFConstituentData** objects reference. If there is no data in the period you request, the history result is empty.

```
PY
def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
    # Get the 20 largest weighting constituents
    selected = sorted([c for c in constituents if c.Weight],
        key=lambda c: c.Weight, reverse=True)[:20]
    selectedSymbols = [c.Symbol for c in selected]

    history = History(selectedSymbols, 10, Resolution.Daily)

    return selectedSymbols
```

Example Applications

The ETF Constituents dataset provides an excellent source of tradable universes for strategies without selection bias. When you use an ETF universe, the original ETF can serve as an excellent benchmark for your strategy performance. Other use cases include the following:

- Creating an index-tracking algorithm for customized passive portfolio management
- Performing statistical arbitrage with the base ETF


Clone

QuantConnect

US Equity Coarse Universe

Introduction

The US Equity Coarse Universe dataset by QuantConnect is a daily universe of all trading stocks in the US for a given day with the end of day price and volume. The data covers 30,000 US Equities in total, with approximately 8,000 Equities per day. The data starts in January 1998 and is delivered each trading day. This dataset is created by taking the closing auction price tick from the daily L1 trade and quote exchange dumps.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Equity Coarse Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US Equity Coarse Universe dataset:

```
from QuantConnect.DataSource import *
self.AddUniverse(self.CoarseSelectionFunction)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	30,000 US Equities
Data Density	Dense
Resolution	Daily
Timezone	New York

Data Point Attributes

The US Equity Coarse Universe dataset provides **CoarseFundamental** objects, which have the following attributes:

Requesting Data

To add US Equity Coarse Universe data to your algorithm, call the **AddUniverse** method with a selection function.

```
PY
class USEquityCoarseUniverseConstituentsDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 7, 1)
        self.SetCash(100000)

        self.AddUniverse(self.CoarseSelectionFunction)
```

Accessing Data

To access US Equity Coarse Universe data, use the **CoarseFundamental** objects in your selection function. The data is available in daily resolution.

```
PY
def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:

    sorted_by_dollar_volume = sorted(coarse, key=lambda x: x.DollarVolume, reverse=True)[:3]
    for cf in sorted_by_dollar_volume:
        self.Debug(f"{cf.EndTime} :: {cf.Symbol} : {cf.AdjustedPrice} :: {cf.DollarVolume}")

    return [ x.Symbol for x in sortedByDollarVolume]
```

Historical Data

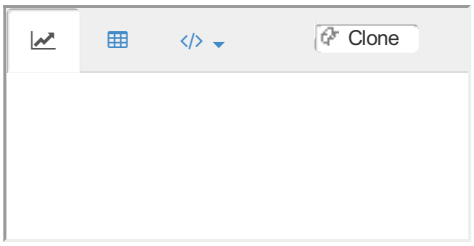
You can't request historical **CoarseFundamental** objects, but you can you can use the **Symbol** member of each object to request historical market data of the securities that the **CoarseFundamental** objects reference. If there is no data in the period you request, the history result is empty.

```
PY
def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:

    selected_symbols = [x.Symbol
        for x in sorted(coarse, key=lambda x: x.DollarVolume, reverse=True)[:3]]

    history = self.History(selected_symbols, 10, Resolution.Daily)

    return selected_symbols
```

Example Applications

The US Security Master dataset enables you to accurately design a universe of US Equities. Examples include the following strategies:

- Selecting securities with the largest dollar volume
- Selecting securities within a specific price range
- Selecting securities that have fundamental data available (requires an additional fine universe selection model)


Clone

QuantConnect

US Equity Security Master

Introduction

The US Equity Security Master dataset by QuantConnect tracks US Equity corporate actions, including splits, dividends, delistings, mergers, and ticker changes through history. The data covers approximately 27,500 US Equities, starts in January 1998, and is delivered on a daily update frequency. You can easily download and install the dataset with the LEAN CLI so it's ready to use by LEAN. LEAN automatically handles all corporate actions and passes them into your algorithm as events .

For more information about the US Equity Security Master dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Data Summary

Data is delivered as a daily updated zip archive of map and factor files. The data is designed to be used in the LEAN Engine and cannot be consumed another way. The following table shows the dataset properties:

Property	Value
Start Date	January 1998
Data Points	Splits, Dividends, Mergers, IPO, & Delistings
Asset Coverage	27,500 US Equities
Resolution	Daily
Timezone	New York

Getting Started

You don't need any special code to utilize the US Equity Security Master. It automatically loads when you request US Equities data .

Supported Assets

To view the supported assets in the US Equity Security Master dataset, see the Data Explorer .

Data Point Attributes

The US Equity Security Master dataset provides **Split** , **Dividend** , **Delisting** , and **SymbolChangedEvent** objects.

Split Attributes

When a split or merger occurs, we pass the previous **Symbol** data into your algorithm. **Split** objects have the following attributes:

Dividend Attributes

Dividend events are triggered on the payment date. **Dividend** objects have the following attributes:

Delisting Attributes

When a security is delisted, we notify your algorithm. **Delisting** objects have the following attributes:

SymbolChangedEvent Attributes

When a security changes their ticker, we notify your algorithm. **SymbolChangedEvent** objects have the following attributes:

Accessing Splits

To get the current split data, index the **Splits** property of the current **Slice** with the Equity **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.Splits.ContainsKey(self.symbol):
        split = slice.Splits[self.symbol]
        split_type = {0: "Warning", 1: "SplitOccurred"}.get(split.Type)
        self.Log(f"Split: {split.Symbol}\t{split.SplitFactor}\t{split.ReferencePrice}\t{split_type}")
```

For more information about accessing splits, see Splits .

Accessing Dividends

To get the current dividend data, index the **Dividends** property of the current **Slice** with the Equity **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.Dividends.ContainsKey(self.symbol):
        dividend = slice.Dividends[self.symbol]
        self.Log(f'Dividend: {dividend.Symbol}\t{dividend.Distribution}\t{dividend.ReferencePrice}')
```

For more information about accessing dividends, see Dividends .

Accessing Delistings

To get the current Delistings data, index the **Delistings** property of the current **Slice** with the Equity **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if slice.Delistings.ContainsKey(self.symbol):
        delisting = slice.Delistings[self.symbol]
        delisting_type = {0: "Warning", 1: "Delisted"}.get(delisting.Type)
        self.Log(f'Delistings: {delisting_type}')
```

For more information about accessing delistings, see Delistings .

Accessing Symbol Change Events

To get the current Symbol change events, index the **SymbolChangedEvents** property of the current **Slice** with the Equity **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

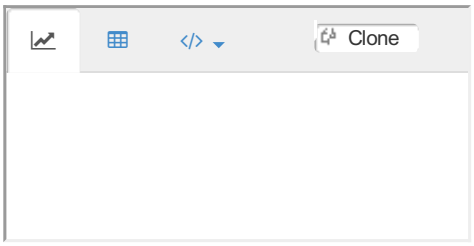
```
def OnData(self, slice: Slice) -> None:
    if slice.SymbolChangedEvents.ContainsKey(self.symbol):
        symbol_changed_event = slice.SymbolChangedEvents[self.symbol]
        self.Log(f"Symbol changed: {symbol_changed_event.OldSymbol} -> {symbol_changed_event.NewSymbol}")
```

For more information about accessing Symbol change events, see Symbol Changes .

Example Applications

The US Security Master enables you to accurately design strategies harnessing any core corporate actions. Examples include the following strategies:

- Post-dividend announcement trading strategies.
- Trading on new Equities by monitoring for IPOs.
- Harnessing split announcements for reverse-split announcement momentum.



QuantConnect

US Futures Security Master

Introduction

The US Futures Security Master dataset by QuantConnect provides mapping reference data for the most liquid contracts of the CME Group exchanges, calculated with popular rolling techniques. The data covers 75 root Future contracts, starts in 2012, and is delivered on a daily frequency with a zip file with all the contract mappings. This dataset is created by daily processing of the US historical Future chains.

This dataset, paired the US Futures dataset, supports the following rolling techniques: ForwardPanamaCanal, BackwardsPanamaCanal, and Backwards Ratio. You can set the specific date of rolling to occur on the LastTradingDay, FirstDayMonth, or on the day where the contract with the greatest OpenInterest changes.

For more information about the US Futures Security Master dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

You don't need any special code to utilize the US Futures Security Master. It automatically loads when you request US Futures data .

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	75 Liquid Futures
Data Density	Regular
Resolution	Daily

Data Point Attributes

The US Futures Security Master dataset provides **SymbolChangedEvent** objects, which have the following attributes:

Key Concept

The US Futures Security Master lets you to construct continuous Futures, allowing the access of normalized historical data of the underlying assets, as well as trading the “lead” Future contracts for those assets.

Continuous Futures refer to sets of rolling lead Future contracts during their actively trading periods. Since Future contracts expire at their maturities, to analyze the historical price of a Future and to apply technical indicators, you need the continuous Future price series.

To access the continuous Future, use the Future's **Symbol** property.

```
PY
future = self.AddFuture(Futures.Energies.CrudeOilWTI,
    dataNormalizationMode = DataNormalizationMode.BackwardsRatio,
    dataMappingMode = DataMappingMode.OpenInterest,
    contractDepthOffset = 0)
self.continuous_contract_symbol = future.Symbol
```

The **dataNormalizationMode** and **dataMappingMode** arguments makes the transition of the underlying contracts seemless. However, the Future **Symbol** doesn't map to an underlying Future contract. It works fine to trade within backtests, but could be subjected to friction costs during live trading since the order price could be a normalized price. For more information about this topic, see the **Live Trading Considerations** section.

Data Normalization Modes

The data normalization mode defines how the price series of two contracts are stitched together when the contract rollovers occur. The **DataNormalizatoinMode** enumeration has the following members available for continuous contracts:

If you use a data normalization mode that's not in the preceding list, LEAN automatically converts it to **DataNormalizationMode.BackwardsRatio** .

Data Mapping Modes

The data mapping mode defines when contract rollovers occur. The **DataMappingMode** enumeration has the following members:

Tracking Contract Changes

As the contracts roll over, the **Mapped** property of the **Future** object references the next contract in the series and you receive a **SymbolChangedEvent** . To get the current **Symbol** change events, index the **SymbolChangedEvents** property of the current **Slice** with the continuous Futures **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your Future at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice)0 -> None:
    if slice.SymbolChangedEvents.ContainsKey(self.continuous_future_symbol):
        symbol_changed_event = slice.SymbolChangedEvents[self.continuous_future_symbol]
        self.Log(f"Symbol changed: {symbolChangedEvent.OldSymbol} -> {symbolChangedEvent.NewSymbol}")
```

SymbolChangedEvent objects have the following attributes:

Live Trading Considerations

You can trade continuous Futures, but the continuous Future **Symbol** doesn't map to a single underlying Future contract. Instead, it represents a set of rolling contracts. Thus, the prices could be frictional during a contract rollover, which could be catastrophic in live trading! For live trading, you should place your orders directly on the underlying contracts. To get the current underlying contract in the continuous Future series, use the **Mapped** property.

```
current_contract = self.continuous_contract.Mapped
self.Buy(current_contract, 1)
```

Data Format

If you download the files in the US Futures Security Master, you get a factor file and a map file for each of the exchanges with supported continuous Futures. To view the files, see the *|data|future|<exchange_name>* directory under your LEAN CLI base directory.

For the factor file, it is a **.zip** collection of REST API styled **.csv** files for each Future **Symbol** , including the date, scaling factors for each type of data normalization and the data mapping mode that indicates the Symbol changing event is on that day for that mapping mode. The following line is an example line in the **.csv** file:

{"Date":"2009-10-31T00:00:00","BackwardsRatioScale":[0.9914163090128755364806866953,1.0,1.0],"BackwardsPanamaCanalScale":[-2.0,0.0,0.0],"ForwardPanamaCanalScale":[0.0,0.0,0.0],"DataMappingMode":1}

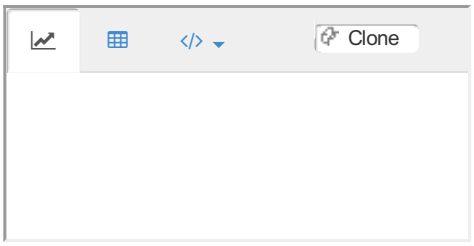
For the map file, it is a **.zip** collection of **.csv** files for each Future **Symbol** , including the date , new underlying contract Symbol, the exchange code, and the data mapping mode that indicates the Symbol changing event is on that day for that mapping mode. The following line is an example line in the **.csv** file:

20091130,aw uii3j0m6zbj9,CBOT,1

Example Applications

The US Futures Security Master enables you to design strategies harnessing continuous Futures contracts. Examples include the following strategies:

- Trading cyclical patterns in commodity Futures.
- Buying gold Futures as an inflation hedge with automatic contract roll overs.
- Detecting arbitrage opportunities between index Futures and Equities.


Clone

Datasets

AlgoSeek

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

US Equities

US Equity Options

US Future Options

US Futures

US Index Options

AlgoSeek

US Equities

Introduction

The US Equities dataset by AlgoSeek is survivorship bias-free daily coverage of every stock traded in the US Securities Information Processors (SIP) CTA/UTP feed since 1998. The dataset covers approximately 27,500 securities, starts in January 1998, and is delivered in any resolution from tick to daily. The Data is collected from the full SIP feed via our Equinix co-located servers, including all trades and quotes published to every exchange as well as FINRA. Over-the-Counter (OTC) trades are not included.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Equities dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

AlgoSeek is the default US Equities dataset on QuantConnect. The following snippet demonstrates how to request data from the US Equities dataset:

```
PY
from QuantConnect.DataSource import *
self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	27,500 US Equities
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	New York

Supported Assets

To view the supported assets in the US Equities dataset, see the Data Explorer .

Data Point Attributes

The US Equities dataset provides **TradeBar** , **QuoteBar** , and **Tick** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

Requesting Data

To add US Equities data to your algorithm, call the **AddEquity** method. Save a reference to the Equity **Symbol** so you can access the data later in your algorithm.

```
class USEquityDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2018, 1, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)
        self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
```

PY

For more information about creating US Equity subscriptions, see Requesting Data or US Equity Universes .

Accessing Data

To get the current US Equities data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Equity **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing US Equities data, see Handling Data .

Historical Data

To get historical US Equity data, call the **History** method with the Equity **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Daily)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Universe Selection

To select a universe of US Equities, see Equity Universes .

Remove Subscriptions

To unsubscribe from a US Equity that you added with the **AddEquity** method, call the **RemoveSecurity** method.

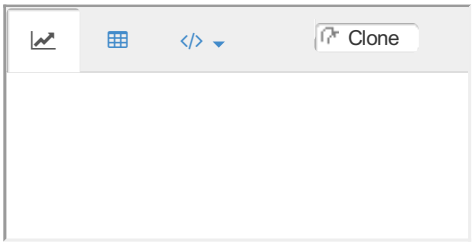
```
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings.

Example Applications

The US Equities dataset enables you to accurately design Equity trading strategies. Examples include the following strategies:

- Momentum strategies using historical returns on the premise that the momentum will continue
- Value strategies using fundamental factors on the premise that the price of undervalued securities will rise
- Factor investing with periodic rebalancing



AlgoSeek

US Equity Options

Introduction

The US Equity Options data by AlgoSeek provides Option data, including prices, strikes, expires, implied volatility, and Greeks. The data covers 4,000 Symbols, starts in January 2012, and is delivered on a minute frequency. This dataset is created by monitoring Options Price Reporting Authority (OPRA) data feed, which consolidates last sale and quotation information originating from the national securities exchanges that have been approved by the Securities and Exchange Commission.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes of the underlying security.

For more information about the US Equity Options dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek was in 2014 with the goal of providing the highest quality, most accurate, ready-to-use data in the financial data industry. AlgoSeek provides access to Equities, ETFs, ETNs, Equity Indices, Equity Options, Futures, and Future Options for quantitative firms and traders.

Getting Started

The following snippet demonstrates how to request data from the US Equity Options dataset:

```py
from QuantConnect.DataSource import *

option = self.AddOption("GOOG")
self.option_symbol = option.Symbol
option.SetFilter(-2, +2, 0, 180)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012*
Asset Coverage	4,000 Symbols
Data Density	Dense
Resolution	Minute, Hourly, & Daily
Timezone	New York

* Some data is available before this date. In 2012, AlgoSeek started to fetch data from 48 OPRA channels instead of 24, increasing the quality of the data.

Supported Assets

To view the supported assets in the US Equity Options dataset, see the Data Explorer .

Data Point Attributes

The US Equity Options dataset provides **TradeBar** , **QuoteBar** , and **OpenInterest** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

OpenInterest Attributes

OpenInterest objects have the following attributes:

Requesting Data

To add US Equity Options data to your algorithm, call the **AddOption** method. Save a reference to the Equity Option **Symbol** so you can access the data later in your algorithm.

```
class USEquityOptionsDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        option = self.AddOption("GOOG")
        self.option_symbol = option.Symbol
        # Set our strike/expiry filter for this option chain
        option.SetFilter(-2, +2, 0, 180)
```

The Equity resolution must be less than or equal to the Equity Option resolution. For example, if you set the Equity resolution to minute, then you must set the Equity Option resolution to minute, hour, or daily.

For more information about creating US Equity Option subscriptions, see Requesting Data or Equity Options Universes .

Accessing Data

To get the current US Equity Options data, index the **OptionChains** property of the current **Slice** with the canonical Equity Option **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your Index Option at every time step. To avoid issues, call the **get** method.

```
def OnData(self, slice: Slice) -> None:
    chain = slice.OptionChains.get(self.option_symbol)
    if chain:
        for contract in chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

You can also iterate through all of the **OptionChain** objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.OptionChains.items():
        for contract in chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

For more information about accessing US Equity Options data, see Handling Data .

Historical Data

You can get historical US Equity Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Equity Options data in an algorithm, call the **History** method with the Equity Option contract **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(contract.Symbol, 100, Resolution.Minute)

# TradeBar objects
history_trade_bars = self.History[TradeBar](contract.Symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](contract.Symbol, 100, Resolution.Minute)
```

For more information about historical data in algorithms, see History Requests .

Historical Data In Research

To get historical US Equity Options data in the Research Environment for an entire Option chain, call the **GetOptionHistory** method with the canonical Option **Symbol** .

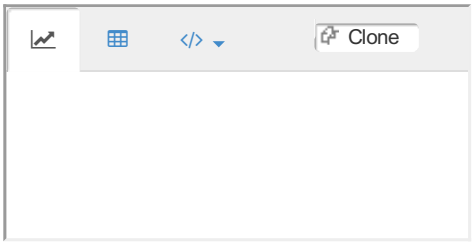
```
qb = QuantBook()
option = qb.AddOption("GOOG")
option.SetFilter(-2, 2, 0, 90)
history = qb.GetOptionHistory(option.Symbol, datetime(2020, 6, 1), datetime(2020, 6, 5))

all_history = history.GetAllData()
expiries = history.GetExpiryDates()
strikes = history.GetStrikes()
```

To get historical data for a single US Equity Option contract, call the **History** method like you would in an algorithm but on the **QuantBook** object. For more information about historical data in the Research Environment, see Equity Options .

Example Applications

The US Equity Options dataset enables you to accurately design Option strategies. Examples include the following strategies:

- Buying put Options to hedge against downward price movement in positive Equity positions
- Exploiting arbitrage opportunities that arise when the price of Option contracts deviate from their theoretical value



AlgoSeek

US Future Options

Introduction

The US Future Options dataset by AlgoSeek provides Option data on US Future contracts, including prices, strikes, expires, implied volatility, and Greeks. The data covers 15 Monthly Future contracts, starts in January 2012, and is delivered on a minute frequency. This dataset is created by monitoring the trading activity on the CME, CBOT, NYMEX, and COMEX markets.

For more information about the US Future Options dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering equities, futures, options, cash forex, and cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

The following snippet demonstrates how to request data from the US Future Options dataset:

```
PY
from QuantConnect.DataSource import *

future = self.AddFuture(Futures.Metals.Gold, Resolution.Minute)
future.SetFilter(0, 90)
self.AddFutureOption(future.Symbol, lambda universe: universe.Strikes(-5, +5))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	15 Monthly Future Contracts
Data Density	Dense
Resolution	Minute, Hourly, & Daily
Timezone	New York

Supported Assets

The following table shows the available Future Options:

Name			
Symbol	**Underlying**	**Market**	**Accessor Code**
Class III Milk Futures			
DC	DC	CME	`Futures.Dairy.ClassIIIMilk`
Crude Oil Futures			
LO	CL	NYMEX	`Futures.Energies.CrudeOilWTI`
NY Harbor ULSD Futures			
OH	HO	NYMEX	`Futures.Energies.HeatingOil`
Henry Hub Natural Gas Futures			
ON	NG	NYMEX	`Futures.Energies.NaturalGas`
RBOB Gasoline Futures			
OB	RB	NYMEX	`Futures.Energies.Gasoline`
U.S. Treasury Bond Futures			
OZB	ZB	CBOT	`Futures.Financials.Y30TreasuryBond`
2-Year T-Note Futures			
OZT	ZT	CBOT	`Futures.Financials.Y2TreasuryNote`
Corn Futures			
OZC	ZC	CBOT	`Futures.Grains.Corn`
Soybean Futures			
OZS	ZS	CBOT	`Futures.Grains.Soybeans`
Chicago SRW Wheat Futures			
OZW	ZW	CBOT	`Futures.Grains.SRWWheat`
E-mini S&P 500 Futures			

Name			
Symbol	**Underlying**	**Market**	**Accessor Code**
ES	ES	CME	`Futures.Indices.SP500EMini`
E-mini Nasdaq-100 Futures			
NQ	NQ	CME	`Futures.Indices.NASDAQ100EMini`
Gold Futures			
OG	GC	COMEX	`Futures.Metals.Gold`
Copper Futures			
HXE	HG	COMEX	`Futures.Metals.Copper`
Silver Futures			
SO	SI	COMEX	`Futures.Metals.Silver`

Data Point Attributes

The US Future Options dataset provides **TradeBar** , **QuoteBar** , and **OpenInterest** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

OpenInterest Attributes

OpenInterest objects have the following attributes:

Requesting Data

To add US Future Options data to your algorithm, call the **AddFutureOption** method.

```
class FutureOptionDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 1, 28)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        future = self.AddFuture(Futures.Metals.Gold, Resolution.Minute)
        future.SetFilter(0, 90)
        self.AddFutureOption(future.Symbol, lambda universe: universe.Strikes(-5, +5))
```

The Future resolution must be less than or equal to the Future Option resolution. For example, if you set the Future

resolution to minute, then the Future Option resolution must be minute, hour, or daily.

For more information about creating Future Options subscriptions, see Requesting Data or Future Options Universes .

Accessing Data

To get the current Future Options data, iterate through the **OptionChains** of the current **Slice** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your Future Options at every time step.

```
PY
def OnData(self, slice: Slice) -> None:
    for canonical_fop_symbol, chain in slice.OptionChains.items():
        for contract in chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

You can also iterate through the **FuturesChains** in the current **Slice** first.

```
PY
def OnData(self, slice: Slice) -> None:
    for continuous_future_symbol, futures_chain in slice.FuturesChains.items():
        # Select a Future Contract and create its canonical FOP Symbol
        futures_contract = [contract for contract in futures_chain][0]
        canonical_fop_symbol = Symbol.CreateCanonicalOption(futures_contract.Symbol)
        option_chain = slice.OptionChains.get(canonical_fop_symbol)
        if option_chain:
            for fop_contract in option_chain:
                self.Log(f"{fop_contract.Symbol} price at {slice.Time}: {fop_contract.LastPrice}")
```

For more information about accessing Future Options data, see Handling Data .

Historical Data

You can get historical US Future Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Future Options data in an algorithm, call the **History** method with the Future Option contract **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(contract.Symbol, 100, Resolution.Minute)

# TradeBar objects
history_trade_bars = self.History[TradeBar](contract.Symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](contract.Symbol, 100, Resolution.Minute)
```

For more information about historical data in algorithms, see History Requests .

Historical Data In Research

To get historical US Future Options data in the Research Environment for the entire Option chain of a Futures contract, call the **GetOptionHistory** method with the Futures contract **Symbol** .

```
qb = QuantBook()
future = qb.AddFuture(Futures.Indices.SP500EMini, Resolution.Minute)
future_symbols = qb.FutureChainProvider.GetFutureContractList(future.Symbol, datetime(2021, 12, 20))
future_contract_symbol = sorted(future_symbols, key=lambda s: s.ID.Date)[0]

start_time = datetime(2021, 12, 1)
end_time = datetime(2021, 12, 10)
option_history = qb.GetOptionHistory(future_contract_symbol, start_time, end_time, Resolution.Minute)

all_history = option_history.GetAllData()
expiries = option_history.GetExpiryDates()
strikes = option_history.GetStrikes()
```

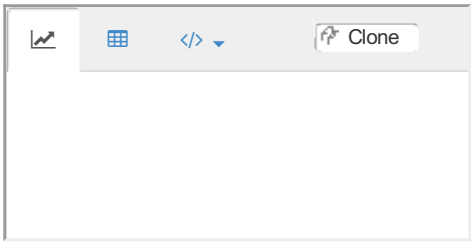
PY

To get historical data for a single US Future Option contract, call the **History** method like you would in an algorithm but on the **QuantBook** object. For more information about historical data in the Research Environment, see Futures Options .

Example Applications

The US Future Options dataset enables you to accurately design Future Option strategies. Examples include the following strategies:

- Selling out of the money Future Option contracts to collect the premium that the Option buyer pays
- Buying put Options to hedge against downward price movement in Future contracts you bought
- Exploiting arbitrage opportunities that arise when the price of Option contracts deviate from their theoretical value



AlgoSeek

US Futures

Introduction

The US Futures dataset by AlgoSeek provides Futures data, including price, volume, open interest, and expiry. The data covers the 75 most liquid contracts, starts in May 2009, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on the CFE, CBOT, NYMEX, ICE*, SGX, India, and COMEX markets.

This dataset does not include ICE Futures, except for Sugar until July 2021.

For more information about the US Futures dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering equities, futures, options, cash forex, and cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

The following snippet demonstrates how to request data from the US Futures dataset:

```
PY
from QuantConnect.DataSource import *

future = self.AddFuture(Futures.Metals.Gold)
future.SetFilter(0, 90)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	May 2009
Asset Coverage	75 Futures
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	• Chicago (CME or CBOT) • New York (NYMEX or COMEX) • Singapore time (SGX) • Indian Standard Time (India)

This dataset includes Sugar ICE Futures until July 2021.

Supported Assets

The following table shows the available Futures:

Name		
Symbol	**Market**	**Accessor Code**
Australian Dollar Futures		
6A	CME	`Futures.Currencies.AUD`
British Pound Futures		
6B	CME	`Futures.Currencies.GBP`
Canadian Dollar Futures		
6C	CME	`Futures.Currencies.CAD`
Euro FX Futures		
6E	CME	`Futures.Currencies.EUR`
Japanese Yen Futures		
6J	CME	`Futures.Currencies.JPY`
Brazilian Real Futures		
6L	CME	`Futures.Currencies.BRL`
Mexican Peso Futures		

Name		
Symbol	**Market**	**Accessor Code**
6M	CME	`Futures.Currencies.MXN`
New Zealand Dollar Futures		
6N	CME	`Futures.Currencies.NZD`
Russian Ruble Futures		
6R	CME	`Futures.Currencies.RUB`
Swiss Franc Futures		
6S	CME	`Futures.Currencies.CHF`
South African Rand Futures		
6Z	CME	`Futures.Currencies.ZAR`
Australian Dollar/Canadian Dollar Futures		
ACD	CME	`Futures.Currencies.AUDCAD`
Australian Dollar/Japanese Yen Futures		
AJY	CME	`Futures.Currencies.AUDJPY`
Australian Dollar/New Zealand Dollar Futures		
ANE	CME	`Futures.Currencies.AUDNZD`
Bitcoin Futures		
BTC	CME	`Futures.Currencies.BTC`
Canadian Dollar/Japanese Yen Futures		
CJY	CME	`Futures.Currencies.CADJPY`
Standard-Size USD/Offshore RMB (CNH) Futures		
CNH	CME	`Futures.Currencies.StandardSizeUSDOffshoreRMBCNH`
US Dollar Index Futures		
DX	ICE	`Futures.Currencies.USD`
E-mini Euro FX Futures		

Name		
Symbol	**Market**	**Accessor Code**
E7	CME	`Futures.Currencies.EuroFXEmini`
Euro/Australian Dollar Futures		
EAD	CME	`Futures.Currencies.EURAUD`
Euro/Canadian Dollar Futures		
ECD	CME	`Futures.Currencies.EURCAD`
Euro/Swedish Krona Futures		
ESK	CME	`Futures.Currencies.EURSEK`
Ether Futures		
ETH	CME	`Futures.Currencies.ETH`
E-mini Japanese Yen Futures		
J7	CME	`Futures.Currencies.JapaneseYenEmini`
Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures		
M6A	CME	`Futures.Currencies.MicroAUD`
Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures		
M6B	CME	`Futures.Currencies.MicroGBP`
Micro USD/CAD Futures		
M6C	CME	`Futures.Currencies.MicroCAD`
Micro Euro/U.S. Dollar (EUR/USD) Futures		
M6E	CME	`Futures.Currencies.MicroEUR`
Micro USD/JPY Futures		
M6J	CME	`Futures.Currencies.MicroUSDJPY`
Micro USD/CHF Futures		
M6S	CME	`Futures.Currencies.MicroUSDCHF`
Micro Bitcoin Futures		

Name		
Symbol	**Market**	**Accessor Code**
MBT	CME	`Futures.Currencies.MicroBTC`
Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures		
MCD	CME	`Futures.Currencies.MicroCADUSD`
Micro Ether Futures		
MET	CME	`Futures.Currencies.MicroEther`
Micro INR/USD Futures		
MIR	CME	`Futures.Currencies.MicroINRUSD`
Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures		
MJY	CME	`Futures.Currencies.MicroJPY`
Micro USD/CNH Futures		
MNH	CME	`Futures.Currencies.MicroUSDCNH`
Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures		
MSF	CME	`Futures.Currencies.MicroCHF`
Cash-settled Butter Futures		
CB	CME	`Futures.Dairy.CashSettledButter`
Cash-Settled Cheese Futures		
CSC	CME	`Futures.Dairy.CashSettledCheese`
Class III Milk Futures		
DC	CME	`Futures.Dairy.ClassIIIMilk`
Dry Whey Futures		
DY	CME	`Futures.Dairy.DryWhey`
Class IV Milk Futures		
GDK	CME	`Futures.Dairy.ClassIVMilk`
Nonfat Dry Milk Futures		

Name		
Symbol	**Market**	**Accessor Code**
GNF	CME	`Futures.Dairy.NonfatDryMilk`
Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures		
1S	NYMEX	`Futures.Energies.PropaneNonLDHMontBelvieu`
Argus Propane Far East Index BALMO Futures		
22	NYMEX	`Futures.Energies.ArgusPropaneFarEastIndexBALMO`
Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures		
A0D	NYMEX	`Futures.Energies.MiniEuropeanThreePointPercentFiveFuelOilBargesPlatts`
Mini Singapore Fuel Oil 180 cst (Platts) Futures		
A0F	NYMEX	`Futures.Energies.MiniSingaporeFuelOil180CstPlatts`
Gulf Coast ULSD (Platts) Up-Down BALMO Futures		
A1L	NYMEX	`Futures.Energies.GulfCoastULSDPlattsUpDownBALMO`
Gulf Coast Jet (Platts) Up-Down BALMO Futures		
A1M	NYMEX	`Futures.Energies.GulfCoastJetPlattsUpDownBALMO`
Propane Non-LDH Mont Belvieu (OPIS) Futures		
A1R	NYMEX	`Futures.Energies.PropaneNonLDHMontBelvieuOPIS`
European Propane CIF ARA (Argus) BALMO Futures		
A32	NYMEX	`Futures.Energies.EuropeanPropaneCIFARAArgusBALMO`
Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures		
A3G	NYMEX	`Futures.Energies.PremiumUnleadedGasoline10ppmFOBMEDPlatts`
Argus Propane Far East Index Futures		
A7E	NYMEX	`Futures.Energies.ArgusPropaneFarEastIndex`

Name		
Symbol	**Market**	**Accessor Code**
Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures		
A7I	NYMEX	`Futures.Energies.GasolineEurobobOxyNWEBargesArgusCrackSpreadBALMO`
Mont Belvieu Natural Gasoline (OPIS) Futures		
A7Q	NYMEX	`Futures.Energies.MontBelvieuNaturalGasolineOPIS`
Mont Belvieu Normal Butane (OPIS) BALMO Futures		
A8J	NYMEX	`Futures.Energies.MontBelvieuNormalButaneOPISBALMO`
Conway Propane (OPIS) Futures		
A8K	NYMEX	`Futures.Energies.ConwayPropaneOPIS`
Mont Belvieu LDH Propane (OPIS) BALMO Futures		
A8O	NYMEX	`Futures.Energies.MontBelvieuLDHPropaneOPISBALMO`
Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures		
A91	NYMEX	`Futures.Energies.ArgusPropaneFarEastIndexVsEuropeanPropaneCIFARAArgus`
Argus Propane (Saudi Aramco) Futures		
A9N	NYMEX	`Futures.Energies.ArgusPropaneSaudiAramco`
Group Three ULSD (Platts) vs. NY Harbor ULSD Futures		
AA6	NYMEX	`Futures.Energies.GroupThreeULSDPlattsVsNYHarborULSD`
Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures		
AA8	NYMEX	`Futures.Energies.GroupThreeSubOctaneGasolinePlattsVsRBOB`
Singapore Fuel Oil 180 cst (Platts) BALMO Futures		
ABS	NYMEX	`Futures.Energies.SingaporeFuelOil180cstPlattsBALMO`

Name		
Symbol	**Market**	**Accessor Code**
Singapore Fuel Oil 380 cst (Platts) BALMO Futures		
ABT	NYMEX	`Futures.Energies.SingaporeFuelOil380cstPlattsBALMO`
Mont Belvieu Ethane (OPIS) Futures		
AC0	NYMEX	`Futures.Energies.MontBelvieuEthaneOPIS`
Mont Belvieu Normal Butane (OPIS) Futures		
AD0	NYMEX	`Futures.Energies.MontBelvieuNormalButaneOPIS`
Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures		
ADB	NYMEX	`Futures.Energies.BrentCrudeOilVsDubaiCrudeOilPlatts`
Argus LLS vs. WTI (Argus) Trade Month Futures		
AE5	NYMEX	`Futures.Energies.ArgusLLSvsWTIArgusTradeMonth`
Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures		
AGA	NYMEX	`Futures.Energies.SingaporeGasoilPlattsVsLowSulphurGasoilFutures`
Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures		
AJL	NYMEX	`Futures.Energies.LosAngelesCARBOBGasolineOPISvsRBOBGasoline`
Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures		
AJS	NYMEX	`Futures.Energies.LosAngelesJetOPISvsNYHarborULSD`
Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures		
AKL	NYMEX	`Futures.Energies.LosAngelesCARBDieselOPISvsNYHarborULSD`
European Naphtha (Platts) BALMO Futures		
AKZ	NYMEX	`Futures.Energies.EuropeanNaphthaPlattsBALMO`
European Propane CIF ARA (Argus) Futures		

Name		
Symbol	**Market**	**Accessor Code**
APS	NYMEX	`Futures.Energies.EuropeanPropaneCIFARAArgus`
Mont Belvieu Natural Gasoline (OPIS) BALMO Futures		
AR0	NYMEX	`Futures.Energies.MontBelvieuNaturalGasolineOPISBALMO`
RBOB Gasoline Crack Spread Futures		
ARE	NYMEX	`Futures.Energies.RBOBGasolineCrackSpread`
Gulf Coast HSFO (Platts) BALMO Futures		
AVZ	NYMEX	`Futures.Energies.GulfCoastHSFOPlattsBALMO`
Mars (Argus) vs. WTI Trade Month Futures		
AYV	NYMEX	`Futures.Energies.MarsArgusVsWTITradeMonth`
Mars (Argus) vs. WTI Financial Futures		
AYX	NYMEX	`Futures.Energies.MarsArgusVsWTIFinancial`
Ethanol T2 FOB Rdam Including Duty (Platts) Futures		
AZ1	NYMEX	`Futures.Energies.EthanolT2FOBRdamIncludingDutyPlatts`
Brent Crude Futures		
B	ICE	`Futures.Energies.BrentCrude`
Mont Belvieu LDH Propane (OPIS) Futures		
B0	NYMEX	`Futures.Energies.MontBelvieuLDHPropaneOPIS`
Gasoline Euro-bob Oxy NWE Barges (Argus) Futures		
B7H	NYMEX	`Futures.Energies.GasolineEurobobOxyNWEBargesArgus`
WTI-Brent Financial Futures		
BK	NYMEX	`Futures.Energies.WTIBrentFinancial`

Name		
Symbol	**Market**	**Accessor Code**
3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures		
BOO	NYMEX	`Futures.Energies.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread1000mt`
Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures		
BR7	NYMEX	`Futures.Energies.GasolineEurobobOxyNWEBargesArgusBALMO`
Brent Last Day Financial Futures		
BZ	NYMEX	`Futures.Energies.BrentLastDayFinancial`
Crude Oil Futures		
CL	NYMEX	`Futures.Energies.CrudeOilWTI`
Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures		
CRB	NYMEX	`Futures.Energies.GulfCoastCBOBGasolineA2PlattsVsRBOBGasoline`
Clearbrook Bakken Sweet (NE2) Monthly Index Futures		
CSW	NYMEX	`Futures.Energies.ClearbrookBakkenSweetCrudeOilMonthlyIndexNetEnergy`
WTI Financial Futures		
CSX	NYMEX	`Futures.Energies.WTIFinancial`
Chicago Ethanol (Platts) Futures		
CU	NYMEX	`Futures.Energies.ChicagoEthanolPlatts`
Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures		
D1N	NYMEX	`Futures.Energies.SingaporeMogas92UnleadedPlattsBrentCrackSpread`
Dubai Crude Oil (Platts) Financial Futures		
DCB	NYMEX	`Futures.Energies.DubaiCrudeOilPlattsFinancial`
Japan C&F Naphtha (Platts) BALMO Futures		

Name		
Symbol	**Market**	**Accessor Code**
E6	NYMEX	`Futures.Energies.JapanCnFNaphthaPlattsBALMO`
Ethanol Futures		
EH	CBOT	`Futures.Energies.Ethanol`
European Naphtha (Platts) Crack Spread Futures		
EN	NYMEX	`Futures.Energies.EuropeanNaphthaPlattsCrackSpread`
European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures		
EPN	NYMEX	`Futures.Energies.EuropeanPropaneCIFARAArgusVsNaphthaCargoesCIFNWEPlatts`
Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures		
EVC	NYMEX	`Futures.Energies.SingaporeFuelOil380cstPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts`
East-West Gasoline Spread (Platts-Argus) Futures		
EWG	NYMEX	`Futures.Energies.EastWestGasolineSpreadPlattsArgus`
East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures		
EWN	NYMEX	`Futures.Energies.EastWestNaphthaJapanCFvsCargoesCIFNWESpreadPlatts`
RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures		
EXR	NYMEX	`Futures.Energies.RBOBGasolineVsEurobobOxyNWEBargesArgusThreeHundredFiftyThousandGallons`
3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures		
FO	NYMEX	`Futures.Energies.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread`
Freight Route TC14 (Baltic) Futures		
FRC	NYMEX	`Futures.Energies.FreightRouteTC14Baltic`

Name		
Symbol	**Market**	**Accessor Code**
1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures		
FSS	NYMEX	`Futures.Energies.OnePercentFuelOilCargoesFOBNWEPlattsVsThreePointFivePercentFuelOilBargesFOBRdamPlatts`
Low Sulfur Gasoil		
G	ICE	`Futures.Energies.LowSulfurGasoil`
Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures		
GCU	NYMEX	`Futures.Energies.GulfCoastHSFOPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts`
WTI Houston Crude Oil Futures		
HCL	NYMEX	`Futures.Energies.WTIHoustonCrudeOil`
Natural Gas (Henry Hub) Last-day Financial Futures		
HH	NYMEX	`Futures.Energies.NaturalGasHenryHubLastDayFinancial`
NY Harbor ULSD Futures		
HO	NYMEX	`Futures.Energies.HeatingOil`
Natural Gas (Henry Hub) Penultimate Financial Futures		
HP	NYMEX	`Futures.Energies.NaturalGasHenryHubPenultimateFinancial`
WTI Houston (Argus) vs. WTI Trade Month Futures		
HTT	NYMEX	`Futures.Energies.WTIHoustonArgusVsWTITradeMonth`
Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures		
M1B	NYMEX	`Futures.Energies.MicroGasoilZeroPointOnePercentBargesFOBARAPlatts`
Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures		

Name		
Symbol	**Market**	**Accessor Code**
M35	NYMEX	`Futures.Energies.MicroEuropeanThreePointFivePercentFuelOilCargoesFOBMedPlatts`
Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures		
M5F	NYMEX	`Futures.Energies.MicroCoalAPIFivefobNewcastleArgusMcCloskey`
Micro Singapore Fuel Oil 380CST (Platts) Futures		
MAF	NYMEX	`Futures.Energies.MicroSingaporeFuelOil380CSTPlatts`
Micro WTI Crude Oil Futures		
MCL	NYMEX	`Futures.Energies.MicroCrudeOilWTI`
Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures		
MEF	NYMEX	`Futures.Energies.MicroEuropeanThreePointFivePercentOilBargesFOBRdamPlatts`
Henry Hub Natural Gas Futures		
NG	NYMEX	`Futures.Energies.NaturalGas`
Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures		
R5O	NYMEX	`Futures.Energies.MicroEuropeanFOBRdamMarineFuelZeroPointFivePercentBargesPlatts`
RBOB Gasoline Futures		
RB	NYMEX	`Futures.Energies.Gasoline`
Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures		
S5O	NYMEX	`Futures.Energies.MicroSingaporeFOBMarineFuelZeroPointFivePercetPlatts`
Micro 10-Year Yield Futures		
10Y	CBOT	`Futures.Financials.MicroY10TreasuryNote`
Micro 2-Year Yield Futures		

Name		
Symbol	**Market**	**Accessor Code**
2YY	CBOT	`Futures.Financials.MicroY2TreasuryBond`
Micro 30-Year Yield Futures		
30Y	CBOT	`Futures.Financials.MicroY30TreasuryBond`
Micro 5-Year Yield Futures		
5YY	CBOT	`Futures.Financials.MicroY5TreasuryBond`
5-Year USD MAC Swap Futures		
F1U	CBOT	`Futures.Financials.FiveYearUSDMACSwap`
Eurodollar Futures		
GE	CME	`Futures.Financials.EuroDollar`
Ultra 10-Year U.S. Treasury Note Futures		
TN	CBOT	`Futures.Financials.UltraTenYearUSTreasuryNote`
Ultra U.S. Treasury Bond Futures		
UB	CBOT	`Futures.Financials.UltraUSTreasuryBond`
U.S. Treasury Bond Futures		
ZB	CBOT	`Futures.Financials.Y30TreasuryBond`
5-Year T-Note Futures		
ZF	CBOT	`Futures.Financials.Y5TreasuryNote`
10-Year T-Note Futures		
ZN	CBOT	`Futures.Financials.Y10TreasuryNote`
2-Year T-Note Futures		

Name		
Symbol	**Market**	**Accessor Code**
ZT	CBOT	`Futures.Financials.Y2TreasuryNote`
Random Length Lumber Futures		
LBS	CME	`Futures.Forestry.RandomLengthLumber`
Black Sea Corn Financially Settled (Platts) Futures		
BCF	CBOT	`Futures.Grains.BlackSeaCornFinanciallySettledPlatts`
Black Sea Wheat Financially Settled (Platts) Futures		
BWF	CBOT	`Futures.Grains.BlackSeaWheatFinanciallySettledPlatts`
KC HRW Wheat Futures		
KE	CBOT	`Futures.Grains.HRWWheat`
Corn Futures		
ZC	CBOT	`Futures.Grains.Corn`
Soybean Oil Futures		
ZL	CBOT	`Futures.Grains.SoybeanOil`
Soybean Meal Futures		
ZM	CBOT	`Futures.Grains.SoybeanMeal`
Oats Futures		
ZO	CBOT	`Futures.Grains.Oats`
Soybean Futures		
ZS	CBOT	`Futures.Grains.Soybeans`
Chicago SRW Wheat Futures		
ZW	CBOT	`Futures.Grains.SRWWheat`
Bloomberg Commodity Index Futures		
AW	CBOT	`Futures.Indices.BloombergCommodityIndex`

Name		
Symbol	**Market**	**Accessor Code**
BankNifty Index		
BANKNIFTY	INDIA	`Futures.Indices.BankNifty`
E-mini Nasdaq-100 Biotechnology Index Futures		
BIO	CME	`Futures.Indices.NASDAQ100BiotechnologyEMini`
E-mini FTSE Emerging Index Futures		
EI	CME	`Futures.Indices.FTSEEmergingEmini`
E-mini S&P MidCap 400 Futures		
EMD	CME	`Futures.Indices.SP400MidCapEmini`
E-mini S&P 500 Futures		
ES	CME	`Futures.Indices.SP500EMini`
S&P-GSCI Commodity Index Futures		
GD	CME	`Futures.Indices.SPGSCICommodity`
USD-Denominated Ibovespa Index Futures		
IBV	CME	`Futures.Indices.USDDenominatedIbovespa`
MSCI Europe NTR		
M1EU	NYSELIFFE	`Futures.Indices.MSCIEuropeNTR`
The MSCI Japan NTR		
M1JP	NYSELIFFE	`Futures.Indices.MSCIJapanNTR`
MSCI Emerging Markets Asia Net Total Return		
M1MSA	NYSELIFFE	`Futures.Indices.MSCIEmergingMarketsAsiaNTR`
Micro E-mini Russell 2000 Index Futures		
M2K	CME	`Futures.Indices.MicroRussell2000EMini`

Name		
Symbol	**Market**	**Accessor Code**
Micro E-mini Standard and Poor's 500 Stock Price Index Futures		
MES	CME	`Futures.Indices.MicroSP500EMini`
Micro E-mini Nasdaq-100 Index Futures		
MNQ	CME	`Futures.Indices.MicroNASDAQ100EMini`
MSCI EAFE Index		
MXEA	NYSELIFFE	`Futures.Indices.MSCIEafeIndex`
MSCI EMERGING MARKETS INDEX		
MXEF	NYSELIFFE	`Futures.Indices.MSCIEmergingMarketsIndex`
MSCI USA Index		
MXUS	NYSELIFFE	`Futures.Indices.MSCIUsaIndex`
Micro E-mini Dow Jones Industrial Average Index Futures		
MYM	CBOT	`Futures.Indices.MicroDow30EMini`
Nifty50 Index		
NIFTY	INDIA	`Futures.Indices.Nifty50`
SGX Nikkei 225 Index Futures		
NK	SGX	`Futures.Indices.Nikkei225Yen`
Nikkei/USD Futures		
NKD	CME	`Futures.Indices.Nikkei225Dollar`
E-mini Nasdaq-100 Futures		
NQ	CME	`Futures.Indices.NASDAQ100EMini`
E-mini Russell 2000 Index Futures		
RTY	CME	`Futures.Indices.Russell2000EMini`

Name		
Symbol	**Market**	**Accessor Code**
BSE S&P Sensex Index		
SENSEX	INDIA	`Futures.Indices.BseSensex`
MSCI Taiwan Index Futures		
TW	SGX	`Futures.Indices.MSCITaiwanIndex`
VIX futures		
VX	CFE	`Futures.Indices.VIX`
E-mini Dow ($5) Futures		
YM	CBOT	`Futures.Indices.Dow30EMini`
Feeder Cattle Futures		
GF	CME	`Futures.Meats.FeederCattle`
Lean Hog Futures		
HE	CME	`Futures.Meats.LeanHogs`
Live Cattle Futures		
LE	CME	`Futures.Meats.LiveCattle`
Aluminum MW U.S. Transaction Premium Platts (25MT) Futures		
AUP	COMEX	`Futures.Metals.AluminumMWUSTransactionPremiumPlatts25MT`
Aluminium European Premium Duty-Paid (Metal Bulletin) Futures		
EDP	COMEX	`Futures.Metals.AluminiumEuropeanPremiumDutyPaidMetalBulletin`
Gold Futures		
GC	COMEX	`Futures.Metals.Gold`
Copper Futures		
HG	COMEX	`Futures.Metals.Copper`
U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures		

Name		
Symbol	**Market**	**Accessor Code**
HRC	NYMEX	`Futures.Metals.USMidwestDomesticHotRolledCoilSteelCRUIndex`
Micro Gold Futures		
MGC	COMEX	`Futures.Metals.MicroGold`
Micro Gold TAS Futures		
MGT	COMEX	`Futures.Metals.MicroGoldTAS`
Palladium Futures		
PA	NYMEX	`Futures.Metals.Palladium`
Micro Palladium Futures		
PAM	NYMEX	`Futures.Metals.MicroPalladium`
Platinum Futures		
PL	NYMEX	`Futures.Metals.Platinum`
Silver Futures		
SI	COMEX	`Futures.Metals.Silver`
Micro Silver Futures		
SIL	COMEX	`Futures.Metals.MicroSilver`
Mini Sized NY Gold Futures		
YG	NYSELIFFE	`Futures.Metals.MiniNYGold`
Mini Sized NY Silver Future		
YI	NYSELIFFE	`Futures.Metals.MiniNYSilver`
Gold 100 Troy Oz		
ZG	NYSELIFFE	`Futures.Metals.Gold100Oz`
CBOT 5000 Oz Silver Futures		
ZI	NYSELIFFE	`Futures.Metals.Silver5000Oz`
Cocoa Futures		

Name		
Symbol	**Market**	**Accessor Code**
CC	ICE	`Futures.Softs.Cocoa`
Cotton No. 2 Futures		
CT	ICE	`Futures.Softs.Cotton2`
Coffee C Arabica Futures		
KC	ICE	`Futures.Softs.Coffee`
Frozen Concentrated Orange Juice		
OJ	ICE	`Futures.Softs.OrangeJuice`
Sugar No. 11 Futures		
SB	ICE	`Futures.Softs.Sugar11`
No. 11 Sugar Futures		
YO	NYMEX	`Futures.Softs.Sugar11CME`

Data Point Attributes

The US Futures dataset provides **FuturesChain** , **Future** , and **OpenInterest** objects. To configure the continuous Future settings, use the **DataNormalizationMode** and **DataMappingMode** enumerations.

DataNormalizationMode Values

The **DataNormalizationMode** enumeration has the following values:

DataMappingMode Values

The **DataMappingMode** enumeration has the following values:

Future Attributes

Future objects have the following attributes:

FuturesChain Attributes

FuturesChain objects have the following attributes:

OpenInterest Attributes

OpenInterest objects have the following attributes:

Requesting Data

To add US Futures data to your algorithm, call the **AddFuture** method. Save a reference to the Future so you can access the data later in your algorithm.

```
class USFuturesDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2013, 12, 20)
        self.SetEndDate(2014, 2, 20)
        self.SetCash(1000000)

        future = self.AddFuture(Futures.Metals.Gold)
        future.SetFilter(0, 90)
        self.future_symbol = future.Symbol
```

For more information about creating Future subscriptions, see Requesting Data or Futures Universes .

Accessing Data

To get the current US Futures data, index the **FuturesChains** property of the current **Slice** with the canonical Futures **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your Future at every time step.

```
def OnData(self, slice: Slice) -> None:
    chain = slice.FuturesChains.get(self.future_symbol)
    if chain:
        for contract in chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

You can also iterate through all of the **FuturesChain** objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.FuturesChains.items():
        for contract in chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

For more information about accessing Futures data, see Handling Data .

Historical Data

You can get historical US Futures data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Futures data in an algorithm, call the **History** method with the canonical Futures **Symbol** or a Futures contract **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame objects
contract_history_df = self.History(contract.Symbol, 100, Resolution.Minute)
continuous_history_df = self.History(self.future_symbol,
    start=self.Time - timedelta(days=15),
    end=self.Time,
    resolution=Resolution.Minute,
    fillForward=False,
    extendedMarketHours=False,
    dataMappingMode=DataMappingMode.OpenInterest,
    dataNormalizationMode=DataNormalizationMode.Raw,
    contractDepthOffset=0)

# TradeBar objects
contract_history_trade_bars = self.History[TradeBar](contract.Symbol, 100, Resolution.Minute)
continous_history_trade_bars = self.History[TradeBar](self.future_symbol, 100, Resolution.Minute)

# QuoteBar objects
contract_history_quote_bars = self.History[QuoteBar](contract.Symbol, 100, Resolution.Minute)
continous_history_quote_bars = self.History[QuoteBar](self.future_symbol, 100, Resolution.Minute)

# Tick objects
contract_history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
continous_history_ticks = self.History[Tick](self.future_symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data in algorithms, see History Requests . For more information about the price adjustments for continuous contracts, see Continous Contracts .

Historical Data In Research

To get historical US Futures data in the Research Environment for an entire Futures chain, call the **GetFutureHistory** method with the canonical Future **Symbol** .

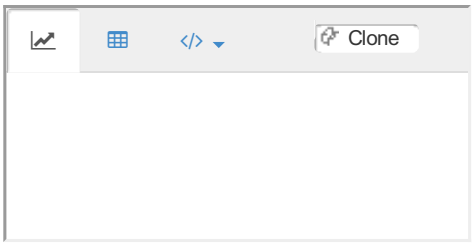
```
qb = QuantBook()
future = qb.AddFuture(Futures.Metals.Gold)
future.SetFilter(0, 90)
history = qb.GetFutureHistory(future.Symbol, datetime(2020, 6, 1), datetime(2020, 6, 5))

all_history = history.GetAllData()
expiries = history.GetExpiryDates()
```

To get historical data for a single US Futures contract or the continuous Futures contract, call the **History** method like you would in an algorithm but on the **QuantBook** object. For more information about historical data in the Research Environment, see Futures .

Example Applications

The US Futures dataset enables you to accurately design Futures strategies. Examples include the following strategies:

- Buying the Futures contract with the most open interest to reduce slippage and market impact
- Trading bull calendar spreads to reduce volatility and margin requirements


Clone

AlgoSeek

US Index Options

Introduction

The US Index Options dataset by AlgoSeek covers European Option contracts for 3 US Indices: SPX, VIX, and NDX. The dataset starts from June 2010 and is delivered on minute resolution. This dataset is created by monitoring the Options Price Reporting Authority (OPRA) data feed, which consolidates last sale and quotation information originating from the national securities exchanges that have been approved by the Securities and Exchange Commission.

For more information about the US Index Options dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

The following snippet demonstrates how to request data from the US Index Options dataset:

```py
from QuantConnect.DataSource import *

self.index_symbol = self.AddIndex('VIX').Symbol
option = self.AddIndexOption(self.index_symbol)
option.SetFilter(-2, 2, 0, 90)
self.option_symbol = option.Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	June 2010
Asset Coverage	7 Index Options
Data Density	Regular
Resolution	Minute, Hourly, & Daily
Timezone	New York

Supported Assets

The following table shows the available Index Options:

Underlying Index	Underlying Ticker	Target Ticker	Standard Contracts	Weekly Contracts	Tradable on Expiry Day
S&P500	VIX		✓		
S&P500	VIX	VIXW		✓	
S&P500	SPX		✓		
S&P500	SPX	SPXW		✓	✓
NASDAQ-100	NDX		✓		
NASDAQ-100	NDX	NDXP	✓		✓
NASDAQ-100	NDX	NQX	✓	✓	✓

For more information about each underlying Index, see Supported Indices .

Data Point Properties

The US Index Options dataset provides **TradeBar** and **QuoteBar** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Requesting Data

To add US Index Options data to your algorithm, call the **AddIndexOption** method. Save a reference to the Index Option **Symbol** so you can access the data later in your algorithm.

```
class IndexOptionsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1);
        self.SetEndDate(2021, 6, 1);
        self.SetCash(1000000);

        self.index_symbol = self.AddIndex('VIX').Symbol

        standard_option = self.AddIndexOption(self.index_symbol)
        standard_option.SetFilter(-2, 2, 0, 90)
        self.standard_option_symbol = standard_option.Symbol

        weekly_option = self.AddIndexOption(self.index_symbol, "VIXW")
        weekly_option.SetFilter(-2, 2, 0, 90)
        self.weekly_option_symbol = weekly_option.Symbol
```

The Index resolution must be less than or equal to the Index Option resolution. For example, if you set the Index resolution to minute, then you must set the Index Option resolution to minute, hour, or daily.

For more information about creating US Index Option subscriptions, see Requesting Data or Index Options Universes .

Accessing Data

To get the current US Index Options data, index the **OptionChains** property of the current **Slice** with the canonical Index Option **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your Index Option at every time step.

```
PY
def OnData(self, slice: Slice) -> None:
    standard_chain = slice.OptionChains.get(self.standard_option_symbol)
    if standard_chain:
        for contract in standard_chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")

    weekly_chain = slice.OptionChains.get(self.weekly_option_symbol)
    if weekly_chain:
        for contract in weekly_chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

You can also iterate through all of the **OptionChain** objects in the current **Slice** .

```
PY
def OnData(self, slice: Slice) -> None:
    for canonical_symbol, chain in slice.OptionChains.items():
        for contract in chain:
            self.Log(f"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}")
```

For more information about accessing US Index Options data, see Handling Data .

Historical Data

You can get historical US Index Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Index Options data in an algorithm, call the **History** method with the Index Option contract **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(contract.Symbol, 100, Resolution.Minute)

# TradeBar objects
history_trade_bars = self.History[TradeBar](contract.Symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](contract.Symbol, 100, Resolution.Minute)
```

For more information about historical data in algorithms, see History Requests .

Historical Data In Research

To get historical US Index Options data in the Research Environment for an entire Option chain, call the

GetOptionHistory method with the canonical Option **Symbol** .

```
qb = QuantBook()
index_symbol = qb.AddIndex('VIX').Symbol
option = qb.AddIndexOption(index_symbol) # or qb.AddIndexOption(index_symbol, "VIXW")
option.SetFilter(-2, 2, 0, 90)
history = qb.GetOptionHistory(option.Symbol, datetime(2020, 6, 1), datetime(2020, 6, 5))

all_history = history.GetAllData()
expiries = history.GetExpiryDates()
strikes = history.GetStrikes()
```

To get historical data for a single US Index Option contract, call the **History** method like you would in an algorithm but on the **QuantBook** object. For more information about historical data in the Research Environment, see Index Options .

Data Point Attributes

The US Index Options dataset provides **TradeBar** and **QuoteBar** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

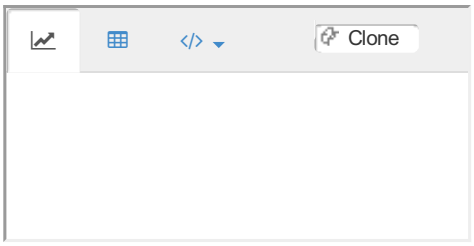
QuoteBar Attributes

QuoteBar objects have the following attributes:

Example Applications

The US Index Options dataset enables you to accurately design strategies for Index Options. Examples include the following strategies:

- Buying VIX call Options to hedge against upcoming volatility
- Buying VIX put Options to capture the natural downward price movement in the VIX index
- Buying SPX put Options to protect against downward price movement in the S&P 500


Clone

Datasets

Morningstar

Morningstar was founded by Joe Mansueto in 1984 with the goal of empowering investors by connecting people to the investing information and tools they need. Morningstar provides access extensive line of products and services for individual investors, financial advisors, asset managers, and retirement plan providers. Morningstar provides data on approximately 525,000 investment offerings including stocks, mutual funds, and similar vehicles, along with real-time global market data on nearly 18 million equities, indexes, futures, options, commodities, and precious metals, in addition to foreign exchange and Treasury markets. Morningstar also offers investment management services through its investment advisory subsidiaries, with $244 billion in assets under advisement or management as of 2021.

US Fundamental Data

Morningstar

US Fundamental Data

Introduction

The US Fundamental Data by Morningstar tracks US Equity fundamentals. The data covers 5,000 US Equities, starts in January 1998, and is delivered on a daily frequency. This dataset is created by using a combination of string matching, Regular Expressions, and Machine Learning to gather the fundamental data published by companies.

For older symbols, the file date is approximated 45 days after the as of date. When a filing date is present on the Morningstar data, it is used. As we are a quant platform, all the data is loaded using "As Original Reported" figures. If there was a mistake reporting the figure, this will not be fixed later. The market typically responds quickly to these initially reported figures. Data is available for multiple periods depending on the property. Periods available include: 1 mo, 2 mo, 3 mo, 6 mo, 9 mo, 12 mo, 1 Yr, 2 Yr, 3 Yr, and 5 Yr. Morningstar symbols cover the NYSE, NASDAQ, AMEX, and BATS exchanges.

In live trading, Morningstar data is delivered to your algorithm at approximately 6 am each day. The majority of the fundamental data update occurs once per month. This includes updates to all of the key information for each security Morningstar supports. On top of this monthly update, there are daily updates of the financial ratios.

As Morningstar data arrives, it updates the master copy and is passed into your algorithm, similar to how TradeBars are fill-forwarded in your data feed. If there have been no updates for a day, you'll receive the same fundamental data.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Fundamental Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Morningstar was founded by Joe Mansueto in 1984 with the goal of empowering investors by connecting people to the investing information and tools they need. Morningstar provides access extensive line of products and services for individual investors, financial advisors, asset managers, and retirement plan providers. Morningstar provides data on approximately 525,000 investment offerings including stocks, mutual funds, and similar vehicles, along with real-time global market data on nearly 18 million equities, indexes, futures, options, commodities, and precious metals, in addition to foreign exchange and Treasury markets. Morningstar also offers investment management services through its investment advisory subsidiaries, with $244 billion in assets under advisement or management as of 2021.

Getting Started

The following snippet demonstrates how to request data from the US Fundamental dataset:

```
from QuantConnect.DataSource import *

self.AddUniverse(self.SelectCoarse, self.SelectFine)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	5,000 US Equities
Corporate Indicators / Tracked Fields	900 Fields
Data Density	Sparse
Resolution	Daily
Timezone	New York

Supported Assets

To view the supported assets in the US Fundamental dataset, see the Data Explorer .

Data Point Attributes

The US Fundamentals dataset provides **FineFundamental** objects. To filter **FineFundamental** objects, you can use the **MorningstarSectorCode** , **MorningstarIndustryGroupCode** , and **MorningstarIndustryCode** enumeration values.

FineFundamental Attributes

FineFundamental objects have the following attributes:

MorningstarSectorCode Enumeration

Sectors are large super categories of data. To access the sector of an Equity, use the **MorningstarSectorCode** property.

```
filtered_fine = [x for x in fine if x.AssetClassification.MorningstarSectorCode ==
MorningstarSectorCode.Technology]
```

The **MorningstarSectorCode** enumeration has the following members:

MorningstarIndustryGroupCode Enumeration

Industry groups are clusters of related industries which tie together. To access the industry group of an Equity, use the **MorningstarIndustryGroupCode** property.

```
filtered_fine = [x for x in fine if x.AssetClassification.MorningstarIndustryGroupCode ==
MorningstarIndustryGroupCode.ApplicationSoftware]
```

The **MorningstarIndustryGroupCode** enumeration has the following members:

MorningstarIndustryCode Enumeration

Industries are the finest level of classification available and are the individual industries according to the Morningstar classification system. To access the industry group of an Equity, use the **MorningstarIndustryCode** property.

```
filtered_fine = [x for x in fine if x.AssetClassification.MorningstarIndustryCode ==
MorningstarIndustryCode.SoftwareApplication]
```

The **MorningstarIndustryCode** enumeration has the following members:

Exchange ID Values

The exchange ID represents the exchange that lists the Equity. To access the exchange ID of an Equity, use the **PrimaryExchangeID** property.

```
filtered_fine = [x for x in fine if x.CompanyReference.PrimaryExchangeID == "NAS"]
```

The exchanges are represented by the following string values:

String Representation	Exchange
NYS	New York Stock Exchange (NYSE)
NAS	NASDAQ
ASE	American Stock Exchange (AMEX)
TSE	Tokyo Stock Exchange
AMS	Amsterdam Internet Exchange
SGO	Santiago Stock Exchange
XMAD	Madrid Stock Exchange
ASX	Australian Securities Exchange
BVMF	B3 (stock exchange)
LON	London Stock Exchange
TKS	Istanbul Stock Exchange Settlement and Custody Bank

String Representation	Exchange
SHG	Shanghai Exchange
LIM	Lima Stock Exchange
FRA	Frankfurt Stock Exchange
JSE	Johannesburg Stock Exchange
MIL	Milan Stock Exchange
TAE	Tel Aviv Stock Exchange
STO	Stockholm Stock Exchange
ETR	Deutsche Boerse Xetra Core
PAR	Paris Stock Exchange
BUE	Buenos Aires Stock Exchange
KRX	Korea Exchange
SWX	SIX Swiss Exchange
PINX	Pink Sheets (OTC)
CSE	Canadian Securities Exchange
PHS	Philippine Stock Exchange
MEX	Mexican Stock Exchange
TAI	Taiwan Stock Exchange
IDX	Indonesia Stock Exchange
OSL	Oslo Stock Exchange
BOG	Colombia Stock Exchange
NSE	National Stock Exchange of India
HEL	Nasdaq Helsinki
MISX	Moscow Exchange
HKG	Hong Kong Stock Exchange
IST	Istanbul Stock Exchange
BOM	Bombay Stock Exchange

String Representation	Exchange
TSX	Toronto Stock Exchange
BRU	Brussels Stock Exchange
BATS	BATS Global Markets
ARCX	NYSE Arca
GREY	Grey Market (OTC)
DUS	Dusseldorf Stock Exchange
BER	Berlin Stock Exchange
ROCO	Taipei Exchange
CNQ	Canadian Trading and Quotation System Inc.
BSP	Bangko Sentral ng Pilipinas
NEOE	NEO Exchange

Requesting Data

To add US Fundamental data to your algorithm, pass a fine universe selection method to the **AddUniverse** or **AddUniverseSelection** method.

```
class MorningStarDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 7, 1)
        self.SetCash(100000)

        self.AddUniverse(self.SelectCoarse, self.SelectFine)
```

Accessing Data

To access fundamental data, use the **FineFundamental** objects in your second selection function or use the **Fundamental** property of the **Equity** objects in your algorithm. The **Symbol** objects you return from the coarse selection function are the **Symbol** objects that comprise the **FineFundamental** objects in your fine fundamental selection function.

```
def SelectCoarse(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    selected = [c for c in coarse if c.HasFundamentalData]
    sorted_by_dollar_volume = sorted(selected, key=lambda c: c.DollarVolume, reverse=True)
    return [ c.Symbol for c in sorted_by_dollar_volume[:20] ]

def SelectFine(self, fine: List[FineFundamental]) -> List[Symbol]:
    sorted_by_pe_ratio = sorted(fine, key=lambda f: f.ValuationRatios.PERatio, reverse=True)
    return [ f.Symbol for f in sorted_by_pe_ratio[:5] ]
```

Many of the MorningStar values are **MultiPeriodField** objects. These objects represent a timespan of data, normally either **OneMonth** , **ThreeMonths** , **SixMonths** , or **TwelveMonths** . To view the objects, see the the auto-generated classes in the LEAN GitHub repository.

To access fundamental data outside of your fine fundamental selection function, use the **Fundamental** property of the **Equity** objects in your algorithm.

```
fundamentals = self.Securities[symbol].Fundamentals
```

PY

Historical Data

It's currently only possible to get historical fundamental data from the Research Environment. To get historical data, call the **GetFundamental** method with the Equity **Symbol** , a **FineFundamental** property, a start date, and an end date. If there is no data in the period you request, the history result is empty.

```
history = qb.GetFundamental(symbol, "ValuationRatios.PERatio", datetime(2021, 1, 1), datetime(2021, 7, 1))
```

PY

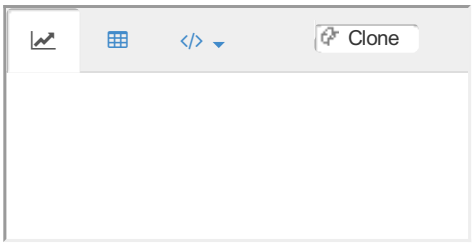
For more information about historical US Equity fundamental data in the Research Environment, see Equity Fundamental Data .

To be notified when historical fundamental data is available in an algorithm, subscribe to GitHub Issue #4890 .

Example Applications

The US Fundamentals dataset enables you to design strategies harnessing fundamental data points. Examples include the following strategies:

- Ranking a universe of securities by a value factor like the book-to-market ratio and buying a subset of the universe with the best factor ranking
- Using the Morningstar asset classification to target specific industries or to ensure your strategy is diversified across several sectors
- Trading securities that recently performed an IPO


Clone

Datasets

TickData

TickData was founded by a futures broker and a programmer in 1984 as the first company in the world to offer historical tick-by-tick prices on the futures and index markets. TickData provides access to comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cash Indices.

US Cash Indices

TickData

US Cash Indices

Introduction

The US Cash Indices dataset by TickData covers 3 US Indices: SPX, VIX, and NDX. The data starts on various dates from Janunary 1998 and is delivered on any frequency from tick to daily. This dataset is created by TickData negotiating directly with exchanges for their official archive and by partnering with real-time data providers who have direct exchange connections and multiple redundant ticker plants.

For more information about the US Cash Indices dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

TickData was founded by a futures broker and a programmer in 1984 as the first company in the world to offer historical tick-by-tick prices on the futures and index markets. TickData provides access to comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cash Indices.

Getting Started

The following snippet demonstrates how to request data from the US Cash Indices dataset:

```
PY
from QuantConnect.DataSource import *
self.symbol = self.AddIndex("VIX", Resolution.Daily).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998*
Coverage	3 US Indices
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	New York

Supported Indices

The following table shows the available indices:

Ticker	Index	Expiry	Start Date
VIX	S&P500	30 Days	Jul 2003
SPX	S&P500	-	Jan 1998
NDX	NASDAQ-100	-	Jan 1998

VIX - CBOE Volatility Index

The Cboe Volatility Index (VIX) is a real-time index that represents the market's expectations for the relative strength of near-term price changes of the S&P 500 index (SPX). Because it's derived from the prices of SPX index Options with near-term expiration dates, it generates a 30-day forward projection of volatility. Volatility, or how fast prices change, is often seen as a way to gauge market sentiment, and in particular, the degree of fear among market participants.

SPX - S&P 500 Index

The S&P 500 Index, or the Standard & Poor's 500 Index, is a market-capitalization-weighted index of the 500 largest publicly-traded companies in the U.S. It is not an exact list of the top 500 U.S. companies by market capitalization because there are other criteria included in the index. The index is widely regarded as the best gauge of large-cap U.S. Equities.

NDX - Nasdaq 100 Index

The Nasdaq-100 Index is a modified market-capitalization-weighted index composed of securities issued by 100 of the largest non-financial companies listed on the Nasdaq Stock Market (Nasdaq). The index includes companies from various industries except for the financial industry, like commercial and investment banks. These non-financial sectors include retail, biotechnology, industrial, technology, health care, and others.

Data Point Attributes

The US Cash Indices dataset provides **TradeBar** and **Tick** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

Requesting Data

To add US Cash Indices data to your algorithm, call the **AddIndex** method. Save a reference to the Index **Symbol** so you can access the data later in your algorithm.

```
class USCashIndexAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddIndex("VIX").Symbol
```

For more information about creating Index subscriptions, see Requesting Data .

Accessing Data

To get the current US Cash Indices data, index the **Bars** or **Ticks** properties of the current **Slice** with the Index **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing US Index data, see Handling Data .

Historical Data

To get historical US Cash Indices data, call the **History** method with the Index **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_bars = self.History[TradeBar](self.symbol, 100, Resolution.Daily)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Remove Subscriptions

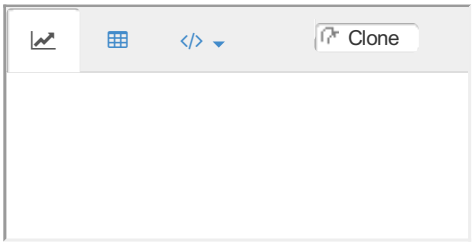
To remove a US Index subscription, call the **RemoveSecurity** method.

```py
self.RemoveSecurity(self.symbol)
```

Example Applications

The US Cash Indices enables you to incorporate popular US indicies into your trading algorithms. Examples include the following use cases:

- Exploring the difference between the Index and the ETF that tracks it
- Using these indicies as the underlying asset for US Index Options strategies
- Understanding the stock market's level of expected forward-looking volatility, also known as the "fear index". When the VIX starts moving higher, it is telling you that traders are getting nervous. When the VIX starts moving lower, it is telling you that traders are gaining confidence.



Datasets

CoinAPI

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Binance Crypto Future Price Data

Binance Crypto Price Data

Binance US Crypto Price Data

Bitfinex Crypto Price Data

Coinbase Crypto Price Data

Kraken Crypto Price Data

CoinAPI

Binance Crypto Future Price Data

Introduction

The Binance Crypto Future Price Data by CoinAPI is for Crypto-currency futures price and volume data points. The data covers 236 Cryptocurrency pairs, starts in August 2020, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Binance.

The Binance Crypto Future Margin Rate Data dataset provides margin interest data to model margin costs.

For more information about the Binance Crypto Future Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Binance Crypto Future Price dataset:

```
PY
def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin)
    self.SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin)

    self.crypto_future_symbol = self.AddCryptoFuture("BTCBUSD", Resolution.Minute).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2020
Asset Coverage	236 Crypto Futures Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC

Supported Assets

The following table shows the available Crypto Future pairs:

Crypto Future Pairs					
1000LUNCBUSD	1000LUNCUSDT	1000SHIBBUSD	1000SHIBUSDT	1000XECUSDT	1INCHUSDT
AAVEUSD	AAVEUSDT	ADABUSD	ADAUSD	ADAUSDT	ALGOUSD
ALGOUSDT	ALICEUSDT	ALPHAUSDT	AMBBUSD	ANCBUSD	ANKRUSDT
ANTUSDT	APEBUSD	APEUSD	APEUSDT	API3USDT	APTBUSD
APTUSD	APTUSDT	ARPAUSDT	ARUSDT	ATAUSDT	ATOMUSD
ATOMUSDT	AUCTIONBUSD	AUDIOUSDT	AVAXBUSD	AVAXUSD	AVAXUSDT
AXSUSD	AXSUSDT	BAKEUSDT	BALUSDT	BANDUSDT	BATUSDT
BCHUSD	BCHUSDT	BELUSDT	BLUEBIRDUSDT	BLZUSDT	BNBBUSD
BNBUSD	BNBUSDT	BNXUSDT	BTCBUSD	BTCDOMUSDT	BTCUSD
BTCUSDT	BTSUSDT	C98USDT	CELOUSDT	CELRUSDT	CHRUSDT
CHZUSD	CHZUSDT	COMPUSDT	COTIUSDT	CRVUSDT	CTKUSDT
CTSIUSDT	CVCUSDT	CVXBUSD	CVXUSDT	DARUSDT	DASHUSDT
DEFIUSDT	DENTUSDT	DGBUSDT	DODOBUSD	DOGEBUSD	DOGEUSD
DOGEUSDT	DOTBUSD	DOTUSD	DOTUSDT	DUSKUSDT	DYDXUSDT
EGLDUSD	EGLDUSDT	ENJUSDT	ENSUSD	ENSUSDT	EOSUSD
EOSUSDT	ETCBUSD	ETCUSD	ETCUSDT	ETHBUSD	ETHUSD
ETHUSDT	FILBUSD	FILUSD	FILUSDT	FLMUSDT	FLOWUSDT
FOOTBALLUSDT	FTMBUSD	FTMUSD	FTMUSDT	FTTBUSD	FTTUSDT
GALABUSD	GALAUSD	GALAUSDT	GALBUSD	GALUSDT	GMTBUSD
GMTUSD	GMTUSDT	GRTUSDT	GTCUSDT	HBARUSDT	HNTUSDT
HOTUSDT	ICPBUSD	ICPUSDT	ICXUSD	ICXUSDT	IMXUSDT
INJUSDT	IOSTUSDT	IOTAUSDT	IOTXUSDT	JASMYUSDT	KAVAUSDT
KLAYUSDT	KNCUSD	KNCUSDT	KSMUSDT	LDOBUSD	LDOUSDT
LEVERBUSD	LINAUSDT	LINKBUSD	LINKUSD	LINKUSDT	LITUSDT
LPTUSDT	LRCUSDT	LTCBUSD	LTCUSD	LTCUSDT	LUNA2BUSD

Crypto Future Pairs					
LUNA2USDT	MANAUSD	MANAUSDT	MASKUSDT	MATICBUSD	MATICUSD
MATICUSDT	MKRUSDT	MTLUSDT	NEARBUSD	NEARUSD	NEARUSDT
NEOUSDT	NKNUSDT	OCEANUSDT	OGNUSDT	OMGUSDT	ONEUSDT
ONTUSDT	OPUSD	OPUSDT	PEOPLEUSDT	PHBBUSD	QNTUSDT
QTUMUSDT	RAYUSDT	REEFUSDT	RENUSDT	RLCUSDT	ROSEUSD
ROSEUSDT	RSRUSDT	RUNEUSD	RUNEUSDT	RVNUSDT	SANDBUSD
SANDUSD	SANDUSDT	SCUSDT	SFPUSDT	SKLUSDT	SNXUSDT
SOLBUSD	SOLUSD	SOLUSDT	SPELLUSDT	SRMUSDT	STGUSDT
STMXUSDT	STORJUSDT	SUSHIUSDT	SXPUSDT	THETAUSD	THETAUSDT
TLMBUSD	TLMUSDT	TOMOUSDT	TRBUSDT	TRXBUSD	TRXUSD
TRXUSDT	UNFIUSDT	UNIBUSD	UNIUSD	UNIUSDT	VETUSD
VETUSDT	WAVESBUSD	WAVESUSDT	WOOUSDT	XEMUSDT	XLMUSD
XLMUSDT	XMRUSD	XMRUSDT	XRPBUSD	XRPUSD	XRPUSDT
XTZUSD	XTZUSDT	YFIUSDT	ZECUSDT	ZENUSDT	ZILUSD
ZILUSDT	ZRXUSDT				

Data Point Attributes

The Binance Crypto Future Price dataset provides **TradeBar** , **QuoteBar** , and **Tick** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

Requesting Data

To add Binance Crypto Future Price data to your algorithm, call the **AddCryptoFuture** method. Save a reference to the Crypto Future **Symbol** so you can access the data later in your algorithm.

```
class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)

        # Set Account Currency to Binance Stable Coin for USD
        self.SetAccountCurrency("BUSD")
        self.SetCash(100000)

        self.SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin)
        self.SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin)

        crypto_future = self.AddCryptoFuture("BTCBUSD", Resolution.Minute)
        # perpetual futures does not have a filter function
        self.symbol = crypto_future.Symbol
```

For more information about creating Crypto Future subscriptions, see Requesting Data .

Accessing Data

To get the current Binance Crypto Future Price data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Crypto Future **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Crypto Future data, see Handling Data .

Historical Data

To get historical Binance Crypto Future Price data, call the **History** method with the Crypto Future **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To unsubscribe from a Crypto Future contract that you added with the **AddCryptoFuture** method, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.symbol)
```

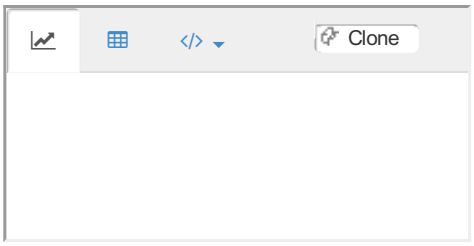
The **RemoveSecurity** method cancels your open orders for the security and liquidates your Crypto Future holdings.

Example Applications

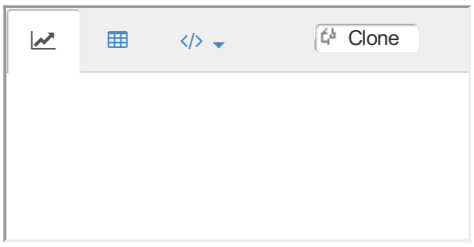
The Binance Crypto Future Price dataset enables you to accurately design strategies for Cryptocurrencies with term structure. Examples include the following strategies:

- Horizontal/Diagonal arbitrage with the underlying cryptocurrencies
- Trade Contango/Backwardation predictions
- Hedge for illiquid cryptocurrencies

Python:



C#:



CoinAPI

Binance Crypto Price Data

Introduction

The Binance Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 1,651 Cryptocurrency pairs, starts in July 2017, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Binance.

For more information about the Binance Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Binance Crypto Price dataset:

```
PY
from QuantConnect.DataSource import *
from QuantConnect.Data.UniverseSelection import *

# Binance accepts both Cash and Margin account types only.
self.SetBrokerageModel(BrokerageName.Binance, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Binance, AccountType.Margin)

self.symbol = self.AddCrypto("BTCBUSD", Resolution.Minute, Market.Binance).Symbol

self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.Binance, self.UniverseSettings,
self.UniverseSelectionFilter))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 2017
Asset Coverage	1,651 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC

Supported Assets

The following table shows the available Cryptocurrency pairs:

Cryptocurrency Pairs					
1INCHBTC	1INCHBUSD	1INCHUSDT	AAVEBKRW	AAVEBNB	AAVEBRL
AAVEBTC	AAVEBUSD	AAVEETH	AAVEUSDT	ACABTC	ACABUSD
ACAUSDT	ACHBTC	ACHBUSD	ACHUSDT	ACMBTC	ACMBUSD
ACMUSDT	ADAAUD	ADABIDR	ADABKRW	ADABNB	ADABRL
ADABTC	ADABUSD	ADAETH	ADAEUR	ADAGBP	ADAPAX
ADARUB	ADATRY	ADATUSD	ADAUSDC	ADAUSDT	ADXBNB
ADXBTC	ADXBUSD	ADXETH	ADXUSDT	AEBNB	AEBTC
AEETH	AERGOBTC	AERGOBUSD	AGIBNB	AGIBTC	AGIETH
AGIXBTC	AGIXBUSD	AGIXUSDT	AGLDBNB	AGLDBTC	AGLDBUSD
AGLDUSDT	AIONBNB	AIONBTC	AIONBUSD	AIONETH	AIONUSDT
AKROBTC	AKROBUSD	AKROUSDT	ALCXBTC	ALCXBUSD	ALCXUSDT
ALGOBIDR	ALGOBNB	ALGOBTC	ALGOBUSD	ALGOETH	ALGOPAX
ALGORUB	ALGOTRY	ALGOTUSD	ALGOUSDC	ALGOUSDT	ALICEBIDR
ALICEBNB	ALICEBTC	ALICEBUSD	ALICETRY	ALICEUSDT	ALPACABNB
ALPACABTC	ALPACABUSD	ALPACAUSDT	ALPHABNB	ALPHABTC	ALPHABUSD
ALPHAUSDT	ALPINEBTC	ALPINEBUSD	ALPINEEUR	ALPINETRY	ALPINEUSDT
AMBBNB	AMBBTC	AMBBUSD	AMBETH	AMBUSDT	AMPBNB
AMPBTC	AMPBUSD	AMPUSDT	ANCBNB	ANCBTC	ANCBUSD
ANCUSDT	ANKRBNB	ANKRBTC	ANKRBUSD	ANKRPAX	ANKRTRY
ANKRTUSD	ANKRUSDC	ANKRUSDT	ANTBNB	ANTBTC	ANTBUSD
ANTUSDT	ANYBTC	ANYBUSD	ANYUSDT	APEAUD	APEBNB
APEBRL	APEBTC	APEBUSD	APEETH	APEEUR	APEGBP
APETRY	APEUSDT	API3BNB	API3BTC	API3BUSD	API3TRY
API3USDT	APPCBNB	APPCBTC	APPCETH	APTBRL	APTBTC

Cryptocurrency Pairs					
APTBUSD	APTETH	APTEUR	APTTRY	APTUSDT	ARBNB
ARBTC	ARBUSD	ARDRBNB	ARDRBTC	ARDRETH	ARDRUSDT
ARKBTC	ARKBUSD	ARKETH	ARNBTC	ARNETH	ARPABNB
ARPABTC	ARPABUSD	ARPAETH	ARPARUB	ARPATRY	ARPAUSDT
ARUSDT	ASRBTC	ASRBUSD	ASRUSDT	ASTBTC	ASTETH
ASTRBTC	ASTRBUSD	ASTRETH	ASTRUSDT	ATABNB	ATABTC
ATABUSD	ATAUSDT	ATMBTC	ATMBUSD	ATMUSDT	ATOMBIDR
ATOMBNB	ATOMBRL	ATOMBTC	ATOMBUSD	ATOMETH	ATOMEUR
ATOMPAX	ATOMTRY	ATOMTUSD	ATOMUSDC	ATOMUSDT	AUCTIONBTC
AUCTIONBUSD	AUCTIONUSDT	AUDBUSD	AUDIOBTC	AUDIOBUSD	AUDIOTRY
AUDIOUSDT	AUDUSDC	AUDUSDT	AUTOBTC	AUTOBUSD	AUTOUSDT
AVABNB	AVABTC	AVABUSD	AVAUSDT	AVAXAUD	AVAXBIDR
AVAXBNB	AVAXBRL	AVAXBTC	AVAXBUSD	AVAXETH	AVAXEUR
AVAXGBP	AVAXTRY	AVAXUSDT	AXSAUD	AXSBIDR	AXSBNB
AXSBRL	AXSBTC	AXSBUSD	AXSETH	AXSTRY	AXSUSDT
BADGERBTC	BADGERBUSD	BADGERUSDT	BAKEBNB	BAKEBTC	BAKEBUSD
BAKEUSDT	BALBNB	BALBTC	BALBUSD	BALUSDT	BANDBNB
BANDBTC	BANDBUSD	BANDUSDT	BARBTC	BARBUSD	BARUSDT
BATBNB	BATBTC	BATBUSD	BATETH	BATPAX	BATTUSD
BATUSDC	BATUSDT	BCCBNB	BCCBTC	BCCETH	BCCUSDT
BCDBTC	BCDETH	BCHABCBTC	BCHABCBUSD	BCHABCPAX	BCHABCTUSD
BCHABCUSDC	BCHABCUSDT	BCHABUSD	BCHBNB	BCHBTC	BCHBUSD
BCHEUR	BCHPAX	BCHSVBTC	BCHSVPAX	BCHSVTUSD	BCHSVUSDC
BCHSVUSDT	BCHTUSD	BCHUSDC	BCHUSDT	BCNBNB	BCNBTC
BCNETH	BCPTBNB	BCPTBTC	BCPTETH	BCPTPAX	BCPTTUSD
BCPTUSDC	BDOTDOT	BEAMBNB	BEAMBTC	BEAMUSDT	BEARBUSD

Cryptocurrency Pairs					
BEARUSDT	BELBNB	BELBTC	BELBUSD	BELETH	BELTRY
BELUSDT	BETABNB	BETABTC	BETABUSD	BETAETH	BETAUSDT
BETHBUSD	BETHETH	BETHUSDT	BGBPUSDC	BICOBTC	BICOBUSD
BICOUSDT	BIFIBNB	BIFIBUSD	BIFIUSDT	BKRWBUSD	BKRWUSDT
BLZBNB	BLZBTC	BLZBUSD	BLZETH	BLZUSDT	BNBAUD
BNBBEARBUSD	BNBBEARUSDT	BNBBIDR	BNBBKRW	BNBBRL	BNBBTC
BNBBULLBUSD	BNBBULLUSDT	BNBBUSD	BNBDAI	BNBETH	BNBEUR
BNBGBP	BNBIDRT	BNBNGN	BNBPAX	BNBRUB	BNBTRY
BNBTUSD	BNBUAH	BNBUSDC	BNBUSDP	BNBUSDS	BNBUSDT
BNBUST	BNBZAR	BNTBTC	BNTBUSD	BNTETH	BNTUSDT
BNXBNB	BNXBTC	BNXBUSD	BNXUSDT	BONDBNB	BONDBTC
BONDBUSD	BONDETH	BONDUSDT	BOTBTC	BOTBUSD	BQXBTC
BQXETH	BRDBNB	BRDBTC	BRDETH	BSWBNB	BSWBUSD
BSWETH	BSWTRY	BSWUSDT	BTCAUD	BTCBBTC	BTCBIDR
BTCBKRW	BTCBRL	BTCBUSD	BTCDAI	BTCEUR	BTCGBP
BTCGYEN	BTCIDRT	BTCNGN	BTCPAX	BTCPLN	BTCRON
BTCRUB	BTCSTBTC	BTCSTBUSD	BTCSTUSDT	BTCTRY	BTCTUSD
BTCUAH	BTCUSDC	BTCUSDP	BTCUSDS	BTCUSDT	BTCUST
BTCVAI	BTCZAR	BTGBTC	BTGBUSD	BTGETH	BTGUSDT
BTSBNB	BTSBTC	BTSBUSD	BTSETH	BTSUSDT	BTTBNB
BTTBRL	BTTBTC	BTTBUSD	BTTCBUSD	BTTCTRY	BTTCUSDC
BTTCUSDT	BTTEUR	BTTPAX	BTTTRX	BTTTRY	BTTTUSD
BTTUSDC	BTTUSDT	BULLBUSD	BULLUSDT	BURGERBNB	BURGERBUSD
BURGERETH	BURGERUSDT	BUSDBIDR	BUSDBKRW	BUSDBRL	BUSDBVND
BUSDDAI	BUSDIDRT	BUSDNGN	BUSDPLN	BUSDRON	BUSDRUB
BUSDTRY	BUSDUAH	BUSDUSDT	BUSDVAI	BUSDZAR	BZRXBNB

Cryptocurrency Pairs					
BZRXBTC	BZRXBUSD	BZRXUSDT	C98BNB	C98BRL	C98BTC
C98BUSD	C98USDT	CAKEAUD	CAKEBNB	CAKEBRL	CAKEBTC
CAKEBUSD	CAKEGBP	CAKEUSDT	CDTBTC	CDTETH	CELOBTC
CELOBUSD	CELOUSDT	CELRBNB	CELRBTC	CELRBUSD	CELRETH
CELRUSDT	CFXBTC	CFXBUSD	CFXUSDT	CHATBTC	CHATETH
CHESSBNB	CHESSBTC	CHESSBUSD	CHESSUSDT	CHRBNB	CHRBTC
CHRBUSD	CHRETH	CHRUSDT	CHZBNB	CHZBRL	CHZBTC
CHZBUSD	CHZEUR	CHZGBP	CHZTRY	CHZUSDT	CITYBNB
CITYBTC	CITYBUSD	CITYUSDT	CKBBTC	CKBBUSD	CKBUSDT
CLOAKBTC	CLOAKETH	CLVBNB	CLVBTC	CLVBUSD	CLVUSDT
CMTBNB	CMTBTC	CMTETH	CNDBNB	CNDBTC	CNDETH
COCOSBNB	COCOSBTC	COCOSBUSD	COCOSTRY	COCOSUSDT	COMPBNB
COMPBTC	COMPBUSD	COMPUSDT	COSBNB	COSBTC	COSBUSD
COSTRY	COSUSDT	COTIBNB	COTIBTC	COTIBUSD	COTIUSDT
COVERBUSD	COVERETH	CREAMBNB	CREAMBUSD	CRVBNB	CRVBTC
CRVBUSD	CRVETH	CRVUSDT	CTKBNB	CTKBTC	CTKBUSD
CTKUSDT	CTSIBNB	CTSIBTC	CTSIBUSD	CTSIUSDT	CTXCBNB
CTXCBTC	CTXCBUSD	CTXCUSDT	CVCBNB	CVCBTC	CVCBUSD
CVCETH	CVCUSDT	CVPBUSD	CVPETH	CVPUSDT	CVXBTC
CVXBUSD	CVXUSDT	DAIBNB	DAIBTC	DAIBUSD	DAIUSDT
DARBNB	DARBTC	DARBUSD	DARETH	DAREUR	DARTRY
DARUSDT	DASHBNB	DASHBTC	DASHBUSD	DASHETH	DASHUSDT
DATABTC	DATABUSD	DATAETH	DATAUSDT	DCRBNB	DCRBTC
DCRBUSD	DCRUSDT	DEGOBTC	DEGOBUSD	DEGOUSDT	DENTBTC
DENTBUSD	DENTETH	DENTTRY	DENTUSDT	DEXEBUSD	DEXEETH

Cryptocurrency Pairs					
DEXEUSDT	DFBUSD	DFETH	DFUSDT	DGBBTC	DGBBUSD
DGBUSDT	DGDBTC	DGDETH	DIABNB	DIABTC	DIABUSD
DIAUSDT	DLTBNB	DLTBTC	DLTETH	DNTBTC	DNTBUSD
DNTETH	DNTUSDT	DOCKBTC	DOCKBUSD	DOCKETH	DOCKUSDT
DODOBTC	DODOBUSD	DODOUSDT	DOGEAUD	DOGEBIDR	DOGEBNB
DOGEBRL	DOGEBTC	DOGEBUSD	DOGEEUR	DOGEGBP	DOGEPAX
DOGERUB	DOGETRY	DOGEUSDC	DOGEUSDT	DOTAUD	DOTBIDR
DOTBKRW	DOTBNB	DOTBRL	DOTBTC	DOTBUSD	DOTETH
DOTEUR	DOTGBP	DOTNGN	DOTRUB	DOTTRY	DOTUSDT
DREPBNB	DREPBTC	DREPBUSD	DREPUSDT	DUSKBNB	DUSKBTC
DUSKBUSD	DUSKPAX	DUSKUSDC	DUSKUSDT	DYDXBNB	DYDXBTC
DYDXBUSD	DYDXETH	DYDXUSDT	EASYBTC	EASYETH	EDOBTC
EDOETH	EGLDBNB	EGLDBTC	EGLDBUSD	EGLDETH	EGLDEUR
EGLDUSDT	ELFBTC	ELFBUSD	ELFETH	ELFUSDT	ENGBTC
ENGETH	ENJBNB	ENJBRL	ENJBTC	ENJBUSD	ENJETH
ENJEUR	ENJGBP	ENJTRY	ENJUSDT	ENSBNB	ENSBTC
ENSBUSD	ENSTRY	ENSUSDT	EOSAUD	EOSBEARBUSD	EOSBEARUSDT
EOSBNB	EOSBTC	EOSBULLBUSD	EOSBULLUSDT	EOSBUSD	EOSETH
EOSEUR	EOSPAX	EOSTRY	EOSTUSD	EOSUSDC	EOSUSDT
EPSBTC	EPSBUSD	EPSUSDT	EPXBUSD	EPXUSDT	ERDBNB
ERDBTC	ERDBUSD	ERDPAX	ERDUSDC	ERDUSDT	ERNBNB
ERNBUSD	ERNUSDT	ETCBNB	ETCBRL	ETCBTC	ETCBUSD
ETCETH	ETCEUR	ETCGBP	ETCPAX	ETCTRY	ETCTUSD
ETCUSDC	ETCUSDT	ETHAUD	ETHBEARBUSD	ETHBEARUSDT	ETHBIDR
ETHBKRW	ETHBRL	ETHBTC	ETHBULLBUSD	ETHBULLUSDT	ETHBUSD

Cryptocurrency Pairs					
ETHDAI	ETHEUR	ETHGBP	ETHNGN	ETHPAX	ETHPLN
ETHRUB	ETHTRY	ETHTUSD	ETHUAH	ETHUSDC	ETHUSDP
ETHUSDT	ETHUST	ETHZAR	EURBUSD	EURUSDT	EVXBTC
EVXETH	EZBTC	EZETH	FARMBNB	FARMBTC	FARMBUSD
FARMETH	FARMUSDT	FETBNB	FETBTC	FETBUSD	FETTRY
FETUSDT	FIDABNB	FIDABTC	FIDABUSD	FIDAUSDT	FILBNB
FILBTC	FILBUSD	FILETH	FILTRY	FILUSDT	FIOBNB
FIOBTC	FIOBUSD	FIOUSDT	FIROBTC	FIROBUSD	FIROETH
FIROUSDT	FISBIDR	FISBRL	FISBTC	FISBUSD	FISTRY
FISUSDT	FLMBNB	FLMBTC	FLMBUSD	FLMUSDT	FLOWBNB
FLOWBTC	FLOWBUSD	FLOWUSDT	FLUXBTC	FLUXBUSD	FLUXUSDT
FORBNB	FORBTC	FORBUSD	FORTHBTC	FORTHBUSD	FORTHUSDT
FORUSDT	FRONTBTC	FRONTBUSD	FRONTETH	FRONTUSDT	FTMAUD
FTMBIDR	FTMBNB	FTMBRL	FTMBTC	FTMBUSD	FTMETH
FTMEUR	FTMPAX	FTMRUB	FTMTRY	FTMTUSD	FTMUSDC
FTMUSDT	FTTBNB	FTTBTC	FTTBUSD	FTTETH	FTTUSDT
FUELBTC	FUELETH	FUNBNB	FUNBTC	FUNETH	FUNUSDT
FXSBTC	FXSBUSD	FXSUSDT	GALAAUD	GALABNB	GALABRL
GALABTC	GALABUSD	GALAETH	GALAEUR	GALATRY	GALAUSDT
GALBNB	GALBRL	GALBTC	GALBUSD	GALETH	GALEUR
GALTRY	GALUSDT	GASBTC	GASBUSD	GBPBUSD	GBPUSDT
GFTBUSD	GHSTBUSD	GHSTETH	GHSTUSDT	GLMBTC	GLMBUSD
GLMETH	GLMRBNB	GLMRBTC	GLMRBUSD	GLMRUSDT	GMTAUD
GMTBNB	GMTBRL	GMTBTC	GMTBUSD	GMTETH	GMTEUR
GMTGBP	GMTTRY	GMTUSDT	GMXBTC	GMXBUSD	GMXUSDT
GNOBNB	GNOBTC	GNOBUSD	GNOUSDT	GNSBTC	GNSUSDT

Cryptocurrency Pairs

GNTBNB	GNTBTC	GNTETH	GOBNB	GOBTC	GRSBTC
GRSETH	GRTBTC	GRTBUSD	GRTETH	GRTEUR	GRTTRY
GRTUSDT	GTCBNB	GTCBTC	GTCBUSD	GTCUSDT	GTOBNB
GTOBTC	GTOBUSD	GTOETH	GTOPAX	GTOTUSD	GTOUSDC
GTOUSDT	GVTBTC	GVTETH	GXSBNB	GXSBTC	GXSETH
GXSUSDT	HARDBNB	HARDBTC	HARDBUSD	HARDUSDT	HBARBNB
HBARBTC	HBARBUSD	HBARUSDT	HCBTC	HCETH	HCUSDT
HEGICBUSD	HEGICETH	HFTBTC	HFTBUSD	HFTUSDT	HIFIETH
HIFIUSDT	HIGHBNB	HIGHBTC	HIGHBUSD	HIGHUSDT	HIVEBNB
HIVEBTC	HIVEBUSD	HIVEUSDT	HNTBTC	HNTBUSD	HNTUSDT
HOOKBNB	HOOKBTC	HOOKBUSD	HOOKUSDT	HOTBNB	HOTBRL
HOTBTC	HOTBUSD	HOTETH	HOTEUR	HOTTRY	HOTUSDT
HSRBTC	HSRETH	ICNBTC	ICNETH	ICPBNB	ICPBTC
ICPBUSD	ICPETH	ICPEUR	ICPRUB	ICPTRY	ICPUSDT
ICXBNB	ICXBTC	ICXBUSD	ICXETH	ICXUSDT	IDEXBNB
IDEXBTC	IDEXBUSD	IDEXUSDT	ILVBNB	ILVBTC	ILVBUSD
ILVUSDT	IMXBNB	IMXBTC	IMXBUSD	IMXUSDT	INJBNB
INJBTC	INJBUSD	INJTRY	INJUSDT	INSBTC	INSETH
IOSTBTC	IOSTBUSD	IOSTETH	IOSTUSDT	IOTABNB	IOTABTC
IOTABUSD	IOTAETH	IOTAUSDT	IOTXBTC	IOTXBUSD	IOTXETH
IOTXUSDT	IQBNB	IQBUSD	IRISBNB	IRISBTC	IRISBUSD
IRISUSDT	JASMYBNB	JASMYBTC	JASMYBUSD	JASMYETH	JASMYEUR
JASMYTRY	JASMYUSDT	JOEBTC	JOEBUSD	JOEUSDT	JSTBNB
JSTBTC	JSTBUSD	JSTUSDT	JUVBTC	JUVBUSD	JUVUSDT
KAVABNB	KAVABTC	KAVABUSD	KAVAETH	KAVAUSDT	KDABTC

Cryptocurrency Pairs

KDABUSD	KDAUSDT	KEEPBNB	KEEPBTC	KEEPBUSD	KEEPUSDT
KEYBTC	KEYBUSD	KEYETH	KEYUSDT	KLAYBNB	KLAYBTC
KLAYBUSD	KLAYUSDT	KMDBTC	KMDBUSD	KMDETH	KMDUSDT
KNCBNB	KNCBTC	KNCBUSD	KNCETH	KNCUSDT	KP3RBNB
KP3RBUSD	KP3RUSDT	KSMAUD	KSMBNB	KSMBTC	KSMBUSD
KSMETH	KSMUSDT	LAZIOBTC	LAZIOBUSD	LAZIOEUR	LAZIOTRY
LAZIOUSDT	LDOBTC	LDOBUSD	LDOUSDT	LENDBKRW	LENDBTC
LENDBUSD	LENDETH	LENDUSDT	LEVERBUSD	LEVERUSDT	LINABNB
LINABTC	LINABUSD	LINAUSDT	LINKAUD	LINKBKRW	LINKBNB
LINKBRL	LINKBTC	LINKBUSD	LINKETH	LINKEUR	LINKGBP
LINKNGN	LINKPAX	LINKTRY	LINKTUSD	LINKUSDC	LINKUSDT
LITBTC	LITBUSD	LITETH	LITUSDT	LOKABNB	LOKABTC
LOKABUSD	LOKAUSDT	LOOMBNB	LOOMBTC	LOOMBUSD	LOOMETH
LPTBNB	LPTBTC	LPTBUSD	LPTUSDT	LQTYBTC	LQTYUSDT
LRCBNB	LRCBTC	LRCBUSD	LRCETH	LRCTRY	LRCUSDT
LSKBNB	LSKBTC	LSKBUSD	LSKETH	LSKUSDT	LTCBNB
LTCBRL	LTCBTC	LTCBUSD	LTCETH	LTCEUR	LTCGBP
LTCNGN	LTCPAX	LTCRUB	LTCTUSD	LTCUAH	LTCUSDC
LTCUSDT	LTOBNB	LTOBTC	LTOBUSD	LTOUSDT	LUNAAUD
LUNABIDR	LUNABNB	LUNABRL	LUNABTC	LUNABUSD	LUNAETH
LUNAEUR	LUNAGBP	LUNATRY	LUNAUSDT	LUNAUST	LUNBTC
LUNCBUSD	LUNCUSDT	LUNETH	MAGICBTC	MAGICBUSD	MAGICUSDT
MANABIDR	MANABNB	MANABRL	MANABTC	MANABUSD	MANAETH
MANATRY	MANAUSDT	MASKBNB	MASKBUSD	MASKUSDT	MATICAUD
MATICBIDR	MATICBNB	MATICBRL	MATICBTC	MATICBUSD	MATICETH

Cryptocurrency Pairs					
MATICEUR	MATICGBP	MATICRUB	MATICTRY	MATICUSDT	MBLBNB
MBLBTC	MBLBUSD	MBLUSDT	MBOXBNB	MBOXBTC	MBOXBUSD
MBOXTRY	MBOXUSDT	MCBNB	MCBTC	MCBUSD	MCOBNB
MCOBTC	MCOETH	MCOUSDT	MCUSDT	MDABTC	MDAETH
MDTBNB	MDTBTC	MDTBUSD	MDTUSDT	MDXBNB	MDXBTC
MDXBUSD	MDXUSDT	MFTBNB	MFTBTC	MFTETH	MFTUSDT
MINABNB	MINABTC	MINABUSD	MINATRY	MINAUSDT	MIRBTC
MIRBUSD	MIRUSDT	MITHBNB	MITHBTC	MITHUSDT	MKRBNB
MKRBTC	MKRBUSD	MKRUSDT	MLNBNB	MLNBTC	MLNBUSD
MLNUSDT	MOBBTC	MOBBUSD	MOBUSDT	MODBTC	MODETH
MOVRBNB	MOVRBTC	MOVRBUSD	MOVRUSDT	MTHBTC	MTHETH
MTLBTC	MTLBUSD	MTLETH	MTLUSDT	MULTIBTC	MULTIBUSD
MULTIUSDT	NANOBNB	NANOBTC	NANOBUSD	NANOETH	NANOUSDT
NASBNB	NASBTC	NASETH	NAVBNB	NAVBTC	NAVETH
NBSBTC	NBSUSDT	NCASHBNB	NCASHBTC	NCASHETH	NEARBNB
NEARBTC	NEARBUSD	NEARETH	NEAREUR	NEARRUB	NEARTRY
NEARUSDT	NEBLBNB	NEBLBTC	NEBLBUSD	NEBLUSDT	NEOBNB
NEOBTC	NEOBUSD	NEOETH	NEOPAX	NEORUB	NEOTRY
NEOTUSD	NEOUSDC	NEOUSDT	NEXOBTC	NEXOBUSD	NEXOUSDT
NKNBNB	NKNBTC	NKNBUSD	NKNUSDT	NMRBTC	NMRBUSD
NMRUSDT	NPXSBTC	NPXSETH	NPXSUSDC	NPXSUSDT	NUAUD
NUBNB	NUBTC	NUBUSD	NULSBNB	NULSBTC	NULSBUSD
NULSETH	NULSUSDT	NURUB	NUUSDT	NXSBNB	NXSBTC
NXSETH	OAXBTC	OAXETH	OCEANBNB	OCEANBTC	OCEANBUSD
OCEANUSDT	OGBTC	OGBUSD	OGNBNB	OGNBTC	OGNBUSD
OGNUSDT	OGUSDT	OMBTC	OMBUSD	OMGBNB	OMGBTC

Cryptocurrency Pairs					
OMGBUSD	OMGETH	OMGUSDT	OMUSDT	ONEBIDR	ONEBNB
ONEBTC	ONEBUSD	ONEETH	ONEPAX	ONETRY	ONETUSD
ONEUSDC	ONEUSDT	ONGBNB	ONGBTC	ONGUSDT	ONTBNB
ONTBTC	ONTBUSD	ONTETH	ONTPAX	ONTTRY	ONTUSDC
ONTUSDT	OOKIBNB	OOKIBUSD	OOKIETH	OOKIUSDT	OPBNB
OPBTC	OPBUSD	OPETH	OPEUR	OPUSDT	ORNBTC
ORNBUSD	ORNUSDT	OSMOBTC	OSMOBUSD	OSMOUSDT	OSTBNB
OSTBTC	OSTETH	OXTBTC	OXTBUSD	OXTUSDT	PAXBNB
PAXBTC	PAXBUSD	PAXETH	PAXGBNB	PAXGBTC	PAXGBUSD
PAXGUSDT	PAXTUSD	PAXUSDT	PEOPLEBNB	PEOPLEBTC	PEOPLEBUSD
PEOPLEETH	PEOPLEUSDT	PERLBNB	PERLBTC	PERLUSDC	PERLUSDT
PERPBTC	PERPBUSD	PERPUSDT	PHABTC	PHABUSD	PHAUSDT
PHBBNB	PHBBTC	PHBBUSD	PHBPAX	PHBTUSD	PHBUSDC
PHBUSDT	PHXBNB	PHXBTC	PHXETH	PIVXBNB	PIVXBTC
PLABNB	PLABTC	PLABUSD	PLAUSDT	PNTBTC	PNTUSDT
POABNB	POABTC	POAETH	POEBTC	POEETH	POLSBNB
POLSBTC	POLSBUSD	POLSUSDT	POLYBNB	POLYBTC	POLYBUSD
POLYUSDT	POLYXBTC	POLYXBUSD	POLYXUSDT	PONDBTC	PONDBUSD
PONDUSDT	PORTOBTC	PORTOBUSD	PORTOEUR	PORTOTRY	PORTOUSDT
POWRBNB	POWRBTC	POWRBUSD	POWRETH	POWRUSDT	PPTBTC
PPTETH	PROMBNB	PROMBTC	PROMBUSD	PROSBUSD	PROSETH
PROSUSDT	PSGBTC	PSGBUSD	PSGUSDT	PUNDIXBUSD	PUNDIXETH
PUNDIXUSDT	PYRBTC	PYRBUSD	PYRUSDT	QIBNB	QIBTC
QIBUSD	QIUSDT	QKCBTC	QKCBUSD	QKCETH	QLCBNB
QLCBTC	QLCETH	QNTBNB	QNTBTC	QNTBUSD	QNTUSDT
QSPBNB	QSPBTC	QSPETH	QTUMBNB	QTUMBTC	QTUMBUSD

Cryptocurrency Pairs					
QTUMETH	QTUMUSDT	QUICKBNB	QUICKBTC	QUICKBUSD	QUICKUSDT
RADBNB	RADBTC	RADBUSD	RADUSDT	RAMPBTC	RAMPBUSD
RAMPUSDT	RAREBNB	RAREBTC	RAREBUSD	RAREUSDT	RAYBNB
RAYBUSD	RAYUSDT	RCNBNB	RCNBTC	RCNETH	RDNBNB
RDNBTC	RDNETH	REEFBIDR	REEFBTC	REEFBUSD	REEFTRY
REEFUSDT	REIBNB	REIBUSD	REIETH	REIUSDT	RENBNB
RENBTC	RENBTCBTC	RENBTCETH	RENBUSD	RENUSDT	REPBNB
REPBTC	REPBUSD	REPUSDT	REQBTC	REQBUSD	REQETH
REQUSDT	RGTBNB	RGTBTC	RGTBUSD	RGTUSDT	RIFBTC
RIFUSDT	RLCBNB	RLCBTC	RLCBUSD	RLCETH	RLCUSDT
RNDRBTC	RNDRBUSD	RNDRUSDT	ROSEBNB	ROSEBTC	ROSEBUSD
ROSEETH	ROSETRY	ROSEUSDT	RPLBTC	RPLBUSD	RPLUSDT
RPXBNB	RPXBTC	RPXETH	RSRBNB	RSRBTC	RSRBUSD
RSRUSDT	RUNEAUD	RUNEBNB	RUNEBTC	RUNEBUSD	RUNEETH
RUNEEUR	RUNEGBP	RUNETRY	RUNEUSDT	RVNBTC	RVNBUSD
RVNTRY	RVNUSDT	SALTBTC	SALTETH	SANDAUD	SANDBIDR
SANDBNB	SANDBRL	SANDBTC	SANDBUSD	SANDETH	SANDTRY
SANDUSDT	SANTOSBRL	SANTOSBTC	SANTOSBUSD	SANTOSTRY	SANTOSUSDT
SCBTC	SCBUSD	SCETH	SCRTBTC	SCRTBUSD	SCRTETH
SCRTUSDT	SCUSDT	SFPBTC	SFPBUSD	SFPUSDT	SHIBAUD
SHIBBRL	SHIBBUSD	SHIBDOGE	SHIBEUR	SHIBGBP	SHIBRUB
SHIBTRY	SHIBUAH	SHIBUSDT	SKLBTC	SKLBUSD	SKLUSDT
SKYBNB	SKYBTC	SKYETH	SLPBIDR	SLPBNB	SLPBUSD
SLPETH	SLPTRY	SLPUSDT	SNGLSBTC	SNGLSETH	SNMBTC
SNMBUSD	SNMETH	SNTBTC	SNTBUSD	SNTETH	SNXBNB
SNXBTC	SNXBUSD	SNXETH	SNXUSDT	SOLAUD	SOLBIDR

Cryptocurrency Pairs					
SOLBNB	SOLBRL	SOLBTC	SOLBUSD	SOLETH	SOLEUR
SOLGBP	SOLRUB	SOLTRY	SOLUSDC	SOLUSDT	SPARTABNB
SPELLBNB	SPELLBTC	SPELLBUSD	SPELLTRY	SPELLUSDT	SRMBIDR
SRMBNB	SRMBTC	SRMBUSD	SRMUSDT	SSVBTC	SSVBUSD
SSVETH	SSVUSDT	STEEMBNB	STEEMBTC	STEEMBUSD	STEEMETH
STEEMUSDT	STGBTC	STGBUSD	STGUSDT	STMXBTC	STMXBUSD
STMXETH	STMXUSDT	STORJBTC	STORJBUSD	STORJETH	STORJTRY
STORJUSDT	STORMBNB	STORMBTC	STORMETH	STORMUSDT	STPTBNB
STPTBTC	STPTBUSD	STPTUSDT	STRATBNB	STRATBTC	STRATBUSD
STRATETH	STRATUSDT	STRAXBTC	STRAXBUSD	STRAXETH	STRAXUSDT
STXBNB	STXBTC	STXBUSD	STXUSDT	SUBBTC	SUBETH
SUNBTC	SUNBUSD	SUNUSDT	SUPERBTC	SUPERBUSD	SUPERUSDT
SUSDBTC	SUSDETH	SUSDUSDT	SUSHIBNB	SUSHIBTC	SUSHIBUSD
SUSHIUSDT	SWRVBNB	SWRVBUSD	SXPAUD	SXPBIDR	SXPBNB
SXPBTC	SXPBUSD	SXPEUR	SXPGBP	SXPTRY	SXPUSDT
SYNBTC	SYNUSDT	SYSBNB	SYSBTC	SYSBUSD	SYSETH
SYSUSDT	TBUSD	TCTBNB	TCTBTC	TCTUSDT	TFUELBNB
TFUELBTC	TFUELBUSD	TFUELPAX	TFUELTUSD	TFUELUSDC	TFUELUSDT
THETABNB	THETABTC	THETABUSD	THETAETH	THETAEUR	THETAUSDT
TKOBIDR	TKOBTC	TKOBUSD	TKOUSDT	TLMBNB	TLMBTC
TLMBUSD	TLMTRY	TLMUSDT	TNBBTC	TNBETH	TNTBTC
TNTETH	TOMOBNB	TOMOBTC	TOMOBUSD	TOMOUSDC	TOMOUSDT
TORNBNB	TORNBTC	TORNBUSD	TORNUSDT	TRBBNB	TRBBTC
TRBBUSD	TRBUSDT	TRIBEBNB	TRIBEBTC	TRIBEBUSD	TRIBEUSDT
TRIGBNB	TRIGBTC	TRIGETH	TROYBNB	TROYBTC	TROYBUSD
TROYUSDT	TRUBTC	TRUBUSD	TRURUB	TRUUSDT	TRXAUD

Cryptocurrency Pairs					
TRXBNB	TRXBTC	TRXBUSD	TRXETH	TRXEUR	TRXNGN
TRXPAX	TRXTRY	TRXTUSD	TRXUSDC	TRXUSDT	TRXXRP
TUSDBNB	TUSDBTC	TUSDBTUSD	TUSDBUSD	TUSDETH	TUSDT
TUSDUSDT	TVKBTC	TVKBUSD	TVKUSDT	TWTBTC	TWTBUSD
TWTTRY	TWTUSDT	UFTBUSD	UFTETH	UMABTC	UMABUSD
UMATRY	UMAUSDT	UNFIBNB	UNFIBTC	UNFIBUSD	UNFIETH
UNFIUSDT	UNIAUD	UNIBNB	UNIBTC	UNIBUSD	UNIETH
UNIEUR	UNIUSDT	USDCBNB	USDCBUSD	USDCPAX	USDCTUSD
USDCUSDT	USDPBUSD	USDPUSDT	USDSBUSDS	USDSBUSDT	USDSPAX
USDSTUSD	USDSUSDC	USDSUSDT	USDTBIDR	USDTBKRW	USDTBRL
USDTBVND	USDTDAI	USDTGYEN	USDTIDRT	USDTNGN	USDTRUB
USDTTRY	USDTUAH	USDTZAR	USTBTC	USTBUSD	USTCBUSD
USTUSDT	UTKBTC	UTKBUSD	UTKUSDT	VENBNB	VENBTC
VENETH	VENUSDT	VETBNB	VETBTC	VETBUSD	VETETH
VETEUR	VETGBP	VETTRY	VETUSDT	VGXBTC	VGXETH
VGXUSDT	VIABNB	VIABTC	VIAETH	VIBBTC	VIBBUSD
VIBEBTC	VIBEETH	VIBETH	VIBUSDT	VIDTBTC	VIDTBUSD
VIDTUSDT	VITEBNB	VITEBTC	VITEBUSD	VITEUSDT	VOXELBNB
VOXELBTC	VOXELBUSD	VOXELETH	VOXELUSDT	VTHOBNB	VTHOBUSD
VTHOUSDT	WABIBNB	WABIBTC	WABIETH	WANBNB	WANBTC
WANETH	WANUSDT	WAVESBNB	WAVESBTC	WAVESBUSD	WAVESETH
WAVESEUR	WAVESPAX	WAVESRUB	WAVESTRY	WAVESTUSD	WAVESUSDC
WAVESUSDT	WAXPBNB	WAXPBTC	WAXPBUSD	WAXPUSDT	WBTCBTC
WBTCBUSD	WBTCETH	WINBNB	WINBRL	WINBTC	WINBUSD
WINEUR	WINGBNB	WINGBTC	WINGBUSD	WINGETH	WINGSBTC
WINGSETH	WINGUSDT	WINTRX	WINUSDC	WINUSDT	WNXMBNB

Cryptocurrency Pairs					
WNXMBTC	WNXMBUSD	WNXMUSDT	WOOBNB	WOOBTC	WOOBUSD
WOOUSDT	WPRBTC	WPRETH	WRXBNB	WRXBTC	WRXBUSD
WRXEUR	WRXUSDT	WTCBNB	WTCBTC	WTCETH	WTCUSDT
XECBUSD	XECUSDT	XEMBNB	XEMBTC	XEMBUSD	XEMETH
XEMUSDT	XLMBNB	XLMBTC	XLMBUSD	XLMETH	XLMEUR
XLMPAX	XLMTRY	XLMTUSD	XLMUSDC	XLMUSDT	XMRBNB
XMRBTC	XMRBUSD	XMRETH	XMRUSDT	XNOBTC	XNOBUSD
XNOETH	XNOUSDT	XRPAUD	XRPBEARBUSD	XRPBEARUSDT	XRPBIDR
XRPBKRW	XRPBNB	XRPBRL	XRPBTC	XRPBULLBUSD	XRPBULLUSDT
XRPBUSD	XRPETH	XRPEUR	XRPGBP	XRPNGN	XRPPAX
XRPRUB	XRPTRY	XRPTUSD	XRPUSDC	XRPUSDT	XTZBNB
XTZBTC	XTZBUSD	XTZETH	XTZTRY	XTZUSDT	XVGBTC
XVGBUSD	XVGETH	XVGUSDT	XVSBNB	XVSBTC	XVSBUSD
XVSUSDT	XZCBNB	XZCBTC	XZCETH	XZCUSDT	XZCXRP
YFIBNB	YFIBTC	YFIBUSD	YFIEUR	YFIIBNB	YFIIBTC
YFIIBUSD	YFIIUSDT	YFIUSDT	YGGBNB	YGGBTC	YGGBUSD
YGGUSDT	YOYOBNB	YOYOBTC	YOYOETH	ZECBNB	ZECBTC
ZECBUSD	ZECETH	ZECPAX	ZECTUSD	ZECUSDC	ZECUSDT
ZENBNB	ZENBTC	ZENBUSD	ZENETH	ZENUSDT	ZILBIDR
ZILBNB	ZILBTC	ZILBUSD	ZILETH	ZILEUR	ZILTRY
ZILUSDT	ZRXBNB	ZRXBTC	ZRXBUSD	ZRXETH	ZRXUSDT

Data Point Attributes

The Binance Crypto Price dataset provides **TradeBar** , **QuoteBar** , **Tick** , and **CryptoCoarseFundamental** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

CryptoCoarseFundamental Attributes

CryptoCoarseFundamental objects have the following attributes:

Requesting Data

To add Binance Crypto Price data to your algorithm, call the **AddCrypto** method. Save a reference to the Crypto **Symbol** so you can access the data later in your algorithm.

```
PY

class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)

        # Set Account Currency to Binance Stable Coin for USD
        self.SetAccountCurrency("BUSD")
        self.SetCash(100000)

        # Binance accepts both Cash and Margin account types.
        self.SetBrokerageModel(BrokerageName.Binance, AccountType.Margin)

        self.symbol = self.AddCrypto("BTCBUSD", Resolution.Minute, Market.Binance).Symbol
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Binance Crypto Price data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Crypto **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Binance Crypto Price data, call the **History** method with the Crypto **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Binance Crypto pairs, call the **AddUniverse** method with a **CryptoCoarseFundamentalUniverse** object. A Crypto coarse universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
from QuantConnect.Data.UniverseSelection import *

def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.Binance, AccountType.Margin)
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.Binance, self.UniverseSettings,
self.UniverseSelectionFilter))

def UniverseSelectionFilter(self, crypto_coarse: List[CryptoCoarseFundamental]) -> List[Symbol]:
    return [c.Symbol for c in crypto_coarse if c.Volume >= 100 and c.VolumeInUsd > 10000]
```

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the **AddCrypto** method, call the **RemoveSecurity** method.

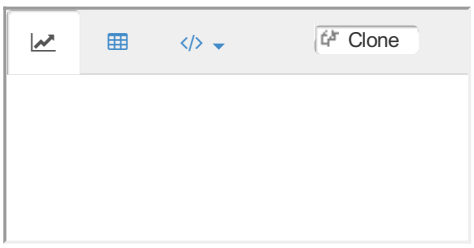
```
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Example Applications

The Binance Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation



CoinAPI

Binance US Crypto Price Data

Introduction

The Binance US Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 174 Cryptocurrency pairs, starts in October 2019, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Binance US.

For more information about the Binance US Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippets demonstrates how to set the brokerage model, request data, and perform universe selection with the Binance US dataset:

```
PY
from QuantConnect.DataSource import *
from QuantConnect.Data.UniverseSelection import *

# Binance US only accepts cash accounts
self.SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash)

self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.BinanceUS).Symbol

self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.BinanceUS, self.UniverseSettings,
self.UniverseSelectionFilter))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 2019
Asset Coverage	174 Cryptocurrency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC

Supported Assets

The following tables shows the available Cryptocurrency pairs:

Cryptocurrency Pairs					
1INCHUSD	1INCHUSD4	1INCHUSDT	AAVEUSD	AAVEUSD4	AAVEUSDT
ACHUSD	ACHUSD4	ACHUSDT	ADABTC	ADABUSD	ADAETH
ADAUSD	ADAUSD4	ADAUSDC	ADAUSDT	ALGOBUSD	ALGOUSD
ALGOUSD4	ALGOUSDT	ALICEUSD	ALICEUSD4	ALICEUSDT	ALPINEUSD
ALPINEUSD4	ALPINEUSDT	AMPUSD	ANKRUSD	ANKRUSD4	ANTUSD
ANTUSD4	ANTUSDT	APEUSD	APEUSD4	APEUSDT	API3USD
API3USD4	API3USDT	APTUSD	APTUSD4	APTUSDT	ASTRUSD
ASTRUSD4	ASTRUSDT	ATOMBTC	ATOMUSD	ATOMUSD4	ATOMUSDT
AUDIOUSD	AUDIOUSD4	AUDIOUSDT	AVAXBTC	AVAXUSD	AVAXUSD4
AVAXUSDT	AXLUSD	AXLUSD4	AXLUSDT	AXSUSD	AXSUSD4
AXSUSDT	BALUSD	BALUSD4	BALUSDT	BANDUSD	BANDUSD4
BANDUSDT	BATUSD	BATUSD4	BATUSDT	BCHBTC	BCHUSD
BCHUSD4	BCHUSDT	BICOUSD	BICOUSD4	BICOUSDT	BNBBTC
BNBBUSD	BNBUSD	BNBUSD4	BNBUSDT	BNTUSD	BNTUSD4
BNTUSDT	BONDUSD	BONDUSD4	BONDUSDT	BOSONUSD	BOSONUSD4
BOSONUSDT	BTCBUSD	BTCDAI	BTCUSD	BTCUSD4	BTCUSDC
BTCUSDT	BTCUST	BTRSTUSD	BTRSTUSD4	BTRSTUSDT	BUSDUSD
BUSDUSD4	BUSDUSDT	CELOUSD	CELOUSD4	CELOUSDT	CELRUSD
CELRUSD4	CELRUSDT	CHZUSD	CHZUSD4	CHZUSDT	CLVUSD
CLVUSD4	CLVUSDT	COMPUSD	COMPUSD4	COMPUSDT	COTIUSD
COTIUSD4	COTIUSDT	CRVUSD	CRVUSD4	CRVUSDT	CTSIUSD
CTSIUSD4	CTSIUSDT	DAIUSD	DAIUSD4	DARUSD	DARUSD4
DARUSDT	DASHUSD	DASHUSD4	DGBUSD	DGBUSD4	DGBUSDT
DIAUSD	DIAUSD4	DIAUSDT	DOGEBTC	DOGEUSD	DOGEUSD4

Cryptocurrency Pairs					
DOGEUSDT	DOTBTC	DOTUSD	DOTUSD4	DOTUSDT	EGLDUSD
EGLDUSD4	EGLDUSDT	ENJUSD	ENJUSD4	ENJUSDT	ENSUSD
ENSUSD4	ENSUSDT	EOSBUSD	EOSUSD	EOSUSD4	EOSUSDT
ETCUSD	ETCUSD4	ETCUSDT	ETHBTC	ETHBUSD	ETHDAI
ETHUSD	ETHUSD4	ETHUSDC	ETHUSDT	FETUSD	FETUSD4
FETUSDT	FILUSD	FILUSD4	FILUSDT	FLOWUSD	FLOWUSD4
FLOWUSDT	FLUXUSD	FLUXUSD4	FLUXUSDT	FORTHUSD	FORTHUSD4
FORTHUSDT	FTMUSD	FTMUSD4	FTMUSDT	GALAUSD	GALAUSD4
GALAUSDT	GALUSD	GALUSD4	GALUSDT	GLMUSD	GLMUSD4
GLMUSDT	GRTUSD	GRTUSD4	GRTUSDT	GTCUSD	GTCUSD4
GTCUSDT	HBARBUSD	HBARUSD	HBARUSD4	HNTUSD	HNTUSD4
HNTUSDT	ICPUSD	ICPUSD4	ICPUSDT	ICXUSD	ICXUSD4
ILVUSD	ILVUSD4	ILVUSDT	IMXUSD	IMXUSD4	IMXUSDT
IOTAUSD	IOTAUSD4	JAMUSD	JAMUSD4	JAMUSDT	JASMYUSD
JASMYUSD4	JASMYUSDT	KAVAUSD	KAVAUSD4	KAVAUSDT	KDAUSD
KDAUSD4	KDAUSDT	KNCUSD	KNCUSD4	KNCUSDT	KSHIBUSD
KSHIBUSD4	KSMUSD	KSMUSD4	KSMUSDT	LAZIOUSD	LAZIOUSD4
LAZIOUSDT	LDOUSD	LDOUSD4	LDOUSDT	LINKBTC	LINKUSD
LINKUSD4	LINKUSDT	LOKAUSD	LOKAUSD4	LOKAUSDT	LOOMUSD
LOOMUSD4	LOOMUSDT	LPTBUSD	LPTUSD	LPTUSD4	LPTUSDT
LRCBTC	LRCUSD	LRCUSD4	LRCUSDT	LSKUSD	LSKUSD4
LSKUSDT	LTCBTC	LTCUSD	LTCUSD4	LTCUSDT	LTOUSD
LTOUSD4	LTOUSDT	MANABTC	MANABUSD	MANAUSD	MANAUSD4
MANAUSDT	MASKUSD	MASKUSD4	MASKUSDT	MATICBTC	MATICBUSD
MATICUSD	MATICUSD4	MATICUSDT	MKRUSD	MKRUSD4	MKRUSDT
MXCUSD	MXCUSD4	MXCUSDT	NANOUSD	NEARBUSD	NEARUSD

Cryptocurrency Pairs					
NEARUSD4	NEARUSDT	NEOUSD	NEOUSD4	NEOUSDT	NMRUSD
NMRUSD4	NMRUSDT	OCEANUSD	OCEANUSD4	OCEANUSDT	OGNUSD
OGNUSD4	OGNUSDT	OMGBUSD	OMGUSD	OMGUSD4	OMGUSDT
ONEBUSD	ONEUSD	ONEUSD4	ONEUSDT	ONTUSD	ONTUSD4
ONTUSDT	OPUSD	OPUSD4	OPUSDT	OXTUSD	OXTUSD4
OXTUSDT	PAXGUSD	PAXGUSD4	PAXGUSDT	POLYBTC	POLYBUSD
POLYUSD	POLYUSDT	POLYXUSD	POLYXUSD4	PONDUSD	PONDUSD4
PONDUSDT	PORTOUSD	PORTOUSD4	PORTOUSDT	PROMUSD	PROMUSD4
PROMUSDT	QNTUSD	QNTUSD4	QNTUSDT	QTUMUSD	QTUMUSD4
QTUMUSDT	RADUSD	RADUSD4	RADUSDT	RAREUSD	RAREUSD4
RAREUSDT	REEFUSD	REEFUSD4	REEFUSDT	RENUSD	RENUSD4
RENUSDT	REPBUSD	REPUSD	REQUSD	REQUSD4	REQUSDT
RLCUSD	RLCUSD4	RLCUSDT	RNDRUSD	RNDRUSD4	RNDRUSDT
ROSEUSD	ROSEUSD4	ROSEUSDT	RVNUSD	RVNUSD4	SANDUSD
SANDUSD4	SANDUSDT	SANTOSUSD	SANTOSUSD4	SANTOSUSDT	SHIBBUSD
SHIBUSD	SHIBUSDT	SKLUSD	SKLUSD4	SKLUSDT	SLPUSD
SLPUSD4	SLPUSDT	SNXUSD	SNXUSD4	SNXUSDT	SOLBTC
SOLBUSD	SOLUSD	SOLUSD4	SOLUSDC	SOLUSDT	SPELLUSD
SPELLUSD4	SPELLUSDT	SRMUSD	SRMUSDT	STGUSD	STGUSD4
STGUSDT	STMXUSD	STMXUSDT	STORJUSD	STORJUSD4	STORJUSDT
SUSHIUSD	SUSHIUSD4	SUSHIUSDT	SYSUSD	SYSUSD4	SYSUSDT
TFUELUSD	TFUELUSD4	TFUELUSDT	THETAUSD	THETAUSD4	THETAUSDT
TLMUSD	TLMUSD4	TLMUSDT	TRXBTC	TRXBUSD	TRXUSD
TRXUSD4	TRXUSDT	TUSD	TUSD4	TUSDT	TUSDUSD
TUSDUSD4	TUSDUSDT	UNIBTC	UNIUSD	UNIUSD4	UNIUSDT
USDCBUSD	USDCUSD	USDCUSD4	USDCUSDT	USDTUSD	USDTUSD4

Cryptocurrency Pairs					
USTUSD	USTUSDT	VETBTC	VETUSD	VETUSD4	VETUSDT
VITEUSD	VITEUSD4	VITEUSDT	VOXELUSD	VOXELUSD4	VOXELUSDT
VTHOUSD	VTHOUSD4	VTHOUSDT	WAVESUSD	WAVESUSD4	WAXPUSD
WAXPUSD4	WAXPUSDT	WBTCBTC	XLMUSD	XLMUSD4	XLMUSDT
XNOUSD	XNOUSD4	XRPBTC	XRPBUSD	XRPUSD	XRPUSDT
XTZBTC	XTZBUSD	XTZUSD	XTZUSD4	YFIUSD	YFIUSD4
YFIUSDT	ZECUSD	ZECUSD4	ZECUSDT	ZENUSD	ZENUSD4
ZENUSDT	ZILBUSD	ZILUSD	ZILUSD4	ZRXUSD	ZRXUSD4
ZRXUSDT					

Data Point Attributes

The Binance US Crypto Price dataset provides **TradeBar** , **QuoteBar** , **Tick** , and **CryptoCoarseFundamental** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

CryptoCoarseFundamental Attributes

CryptoCoarseFundamental objects have the following attributes:

Requesting Data

To add Binance US Crypto Price data to your algorithm, call the **AddCrypto** method. Save a reference to the Crypto **Symbol** so you can access the data later in your algorithm.

```
class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        # BinanceUS accepts Cash account type only, AccountType.Margin will result in an exception.
        self.SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash)

        self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.BinanceUS).Symbol
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Binance US Crypto Price data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Crypto **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Binance US Crypto Price data, call the **History** method with the Crypto **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Binance US Crypto pairs, call the **AddUniverse** method with a **CryptoCoarseFundamentalUniverse** object. A Crypto coarse universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
from QuantConnect.Data.UniverseSelection import *

def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash)
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.BinanceUS, self.UniverseSettings,
self.UniverseSelectionFilter))

def UniverseSelectionFilter(self, crypto_coarse: List[CryptoCoarseFundamental]) -> List[Symbol]:
    return [c.Symbol for c in crypto_coarse if c.Volume >= 100 and c.VolumeInUsd > 10000]
```

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the **AddCrypto** method, call the **RemoveSecurity** method.

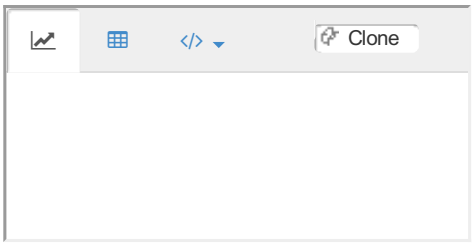
```
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Example Applications

The Binance US Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation


Clone

CoinAPI

Bitfinex Crypto Price Data

Introduction

The Bitfinex Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 309 Cryptocurrency pairs, starts in January 2013, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Bitfinex.

For more information about the Bitfinex Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Bitfinex Crypto Price dataset:

```
PY
from QuantConnect.DataSource import *
from QuantConnect.Data.UniverseSelection import *

# Bitfinex accepts both Cash and Margin type account.
self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin)

self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.Bitfinex).Symbol

self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.Bitfinex, self.UniverseSettings,
self.UniverseSelectionFilter))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2013
Asset Coverage	309 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC

Supported Assets

The following table shows the available Cryptocurrency pairs:

Cryptocurrency Pairs					
1INCHUSD	1INCHUSDT	AAVEUSD	AAVEUSDT	ADABTC	ADAUSD
ADAUSDT	ALBTUSD	ALBTUSDT	ALGOBTC	ALGOUSD	ALGOUSDT
AMPLBTC	AMPLUSD	AMPLUSDT	ANTBTC	ANTETH	ANTUSD
ATOMBTC	ATOMETH	ATOMUSD	AVAXUSD	AVAXUSDT	B21USD
B21USDT	BALUSD	BALUSDT	BANDUSD	BANDUSDT	BATBTC
BATETH	BATUSD	BCHABCUSD	BCHNUSD	BESTUSD	BFTUSD
BMIUSD	BMIUSDT	BNTUSD	BOSONUSD	BOSONUSDT	BSVBTC
BSVUSD	BTCCNHT	BTCEUR	BTCGBP	BTCJPY	BTCUSD
BTCUSDT	BTCXCHF	BTGBTC	BTGUSD	BTSEUSD	BTTUSD
CELUSD	CELUSDT	CHEXUSD	CHZUSD	CHZUSDT	CLOUSD
CNHCNHT	COMPUSD	COMPUSDT	CTKUSD	CTKUSDT	DAIBTC
DAIETH	DAIUSD	DATABTC	DATAUSD	DGBUSD	MDOGEBTC
DOGEUSD	DOGEUSDT	MDOGEUSD	MDOGEUSDT	DOTBTC	DOTUSD
DOTUSDT	DASHBTC	DASHUSD	DUSKBTC	DUSKUSD	PNTBTC
PNTETH	PNTUSD	EGLDUSD	EGLDUSDT	ENJUSD	EOSBTC
EOSDTUSD	EOSDTUSDT	EOSETH	EOSEUR	EOSGBP	EOSJPY
EOSUSD	EOSUSDT	ESSUSD	ETCBTC	ETCUSD	ETH2ETH
ETH2USD	ETH2USDT	ETHBTC	ETHEUR	ETHGBP	ETHJPY
ETHUSD	ETHUSDT	ETPBTC	ETPETH	ETPUSD	EURSUSD
EURTEUR	EURTUSD	EURTUSDT	EXRDBTC	EXRDUSD	FCLUSD
FCLUSDT	FETUSD	FETUSDT	FILUSD	FILUSDT	FORTHUSD
FORTHUSDT	FTMUSD	FTMUSDT	FTTUSD	FTTUSDT	FUNUSD
GNOUSD	GLMUSD	GOTEUR	GOTUSD	GRTUSD	GRTUSDT
GTXUSD	GTXUSDT	HEZUSD	HEZUSDT	ICEUSD	ICPBTC

Cryptocurrency Pairs					
ICPUSD	ICPUSDT	IDUSD	IDUSDT	IOTABTC	IOTAETH
IOTAEUR	IOTAGBP	IOTAJPY	IOTAUSD	IQXUSD	IQXUSDT
JSTBTC	JSTUSD	JSTUSDT	KANUSD	KANUSDT	KNCBTC
KNCUSD	KSMUSD	KSMUSDT	LEOBTC	LEOEOS	LEOETH
LEOUSD	LEOUSDT	LINKUSD	LINKUSDT	LRCUSD	LTCBTC
LTCUSD	LTCUSDT	LUNAUSD	LUNAUSDT	LYMUSD	MIRUSD
MIRUSDT	MKRUSD	MLNUSD	MANABTC	MANAUSD	MOBUSD
MOBUSDT	NEARUSD	NEARUSDT	NECUSD	NEOBTC	NEOETH
NEOEUR	NEOGBP	NEOJPY	NEOUSD	NEXOBTC	NEXOUSD
NEXOUSDT	OCEANUSD	OCEANUSDT	ODEUSD	OMGBTC	OMGETH
OMGUSD	OMNIUSD	ORSUSD	OXYUSD	OXYUSDT	PASSUSD
PAXUSD	PAXUSDT	PLANETSUSD	PLANETSUSDT	PLUUSD	PNKETH
PNKUSD	QASHUSD	QTFBTC	QTFUSD	QTUMBTC	QTUMUSD
RBTCBTC	RBTCUSD	REP2BTC	REP2USD	REQUSD	RINGXUSD
RRBUSD	RRBUSDT	RRTUSD	SANBTC	SANETH	SANUSD
SNGLSUSD	SNTUSD	SNXUSD	SNXUSDT	SOLUSD	SOLUSDT
STORJUSD	SUKUUSD	SUKUUSDT	SUNUSD	SUNUSDT	SUSHIUSD
SUSHIUSDT	TERRAUSTUSD	TERRAUSTUSDT	TESTBTCTESTUSD	TESTBTCTESTUSDT	TRXBTC
TRXETH	TRXEUR	TRXUSD	TUSDUSD	TUSDUSDT	USDCUSD
USDCUSDT	UNIUSD	UNIUSDT	UOPUSD	UOPUSDT	UOSBTC
UOSUSD	USDTCNHT	USDTUSD	UTKUSD	VEEUSD	VELOUSD
VELOUSDT	VETBTC	VETUSD	VSYSBTC	VSYSUSD	WAXUSD
WBTCUSD	XAUTBTC	XAUTUSD	XAUTUSDT	XCHFUSD	XDCUSD
XDCUSDT	XLMBTC	XLMETH	XLMUSD	XLMUSDT	XMRBTC
XMRUSD	XMRUSDT	XRAUSD	XRPBTC	XRPUSD	XRPUSDT
XSNUSD	XTZBTC	XTZUSD	XVGUSD	YFIUSD	YFIUSDT

Cryptocurrency Pairs					
MCSUSD	ZCNUSD	ZECBTC	ZECUSD	ZILBTC	ZILUSD
ZRXBTC	ZRXETH	ZRXUSD			

Data Point Attributes

The Bitfinex Crypto Price dataset provides **TradeBar** , **QuoteBar** , **Tick** , and **CryptoCoarseFundamental** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

CryptoCoarseFundamental Attributes

CryptoCoarseFundamental objects have the following attributes:

Requesting Data

To add Bitfinex Crypto Price data to your algorithm, call the **AddCrypto** method. Save a reference to the Crypto **Symbol** so you can access the data later in your algorithm.

```
class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        # Bitfinex accepts both Cash and Margin type account.
        self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin)

        self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.Bitfinex).Symbol
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Bitfinex Crypto Price data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Crypto **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Bitfinex Crypto Price data, call the **History** method with the Crypto **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Bitfinex Crypto pairs, call the **AddUniverse** method with a **CryptoCoarseFundamentalUniverse** object. A Crypto coarse universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```py
from QuantConnect.Data.UniverseSelection import *

def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin)
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.Kraken, self.UniverseSettings,
self.UniverseSelectionFilter))

def UniverseSelectionFilter(self, crypto_coarse: List[CryptoCoarseFundamental]) -> List[Symbol]:
    return [c.Symbol for c in crypto_coarse if c.Volume >= 100 and c.VolumeInUsd > 10000]
```

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the **AddCrypto** method, call the **RemoveSecurity** method.

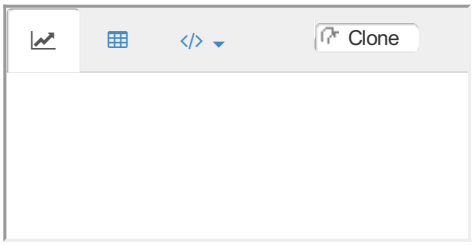
```py
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Example Applications

The Bitfinex Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation



CoinAPI

Coinbase Crypto Price Data

Introduction

The Coinbase Crypto Price Data by CoinAPI provides Cryptocurrency price and volume data points. The data covers 101 Cryptocurrency pairs, starts in January 2015, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Coinbase.

For more information about the Coinbase Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Coinbase Crypto Price dataset:

```
PY
from QuantConnect.DataSource import *
from QuantConnect.Data.UniverseSelection import *

# Coinbase only accepts Cash account type
self.SetBrokerageModel(BrokerageName.GDAX, AccountType.Cash)

self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.GDAX).Symbol

self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.GDAX, self.UniverseSettings,
self.UniverseSelectionFilter))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2015
Asset Coverage	101 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC

Supported Assets

The following table shows the available Cryptocurrency pairs:

Cryptocurrency Pairs					
ALGOEUR	ALGOGBP	ALGOUSD	ATOMBTC	ATOMUSD	BALBTC
BALUSD	BANDBTC	BANDEUR	BANDGBP	BANDUSD	BATETH
BATUSDC	BCHBTC	BCHEUR	BCHGBP	BCHUSD	BTCEUR
BTCGBP	BTCUSD	BTCUSDC	CGLDBTC	CGLDEUR	CGLDGBP
CGLDUSD	COMPBTC	COMPUSD	CVCUSDC	DAIUSD	DAIUSDC
DASHBTC	DASHUSD	DNTUSDC	EOSBTC	EOSEUR	EOSUSD
ETCBTC	ETCEUR	ETCGBP	ETCUSD	ETHBTC	ETHDAI
ETHEUR	ETHGBP	ETHUSD	ETHUSDC	GNTUSDC	KNCBTC
KNCUSD	LINKETH	LINKEUR	LINKGBP	LINKUSD	LOOMUSDC
LRCUSD	LTCBTC	LTCEUR	LTCGBP	LTCUSD	MANAUSDC
MKRBTC	MKRUSD	NMRBTC	NMREUR	NMRGBP	NMRUSD
OMGBTC	OMGEUR	OMGGBP	OMGUSD	OXTUSD	RENBTC
RENUSD	REPBTC	REPUSD	UMABTC	UMAEUR	UMAGBP
UMAUSD	UNIUSD	USDCEUR	USDCGBP	WBTCBTC	WBTCUSD
XLMBTC	XLMEUR	XLMUSD	XRPBTC	XRPEUR	XRPGBP
XRPUSD	XTZBTC	XTZEUR	XTZGBP	XTZUSD	YFIUSD
ZECBTC	ZECUSDC	ZRXBTC	ZRXEUR	ZRXUSD	

Data Point Attributes

The Coinbase Crypto Price dataset provides **TradeBar** , **QuoteBar** , **Tick** , and **CryptoCoarseFundamental** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

CryptoCoarseFundamental Attributes

CryptoCoarseFundamental objects have the following attributes:

Requesting Data

To add Coinbase Crypto Price data to your algorithm, call the **AddCrypto** method. Save a reference to the Crypto **Symbol** so you can access the data later in your algorithm.

```
PY
class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        # Coinbase accepts Cash account type only, AccountType.Margin will result in an exception.
        self.SetBrokerageModel(BrokerageName.GDAX, AccountType.Cash)

        self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.GDAX).Symbol
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Coinbase Crypto Price data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Crypto **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Coinbase Crypto Price data, call the **History** method with the Crypto **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY

# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Coinbase Crypto pairs, call the **AddUniverse** method with a **CryptoCoarseFundamentalUniverse** object. A Crypto coarse universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
PY

from QuantConnect.Data.UniverseSelection import *

def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.GDAX, AccountType.Cash)
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.GDAX, self.UniverseSettings,
self.UniverseSelectionFilter))

def UniverseSelectionFilter(self, crypto_coarse: List[CryptoCoarseFundamental]) -> List[Symbol]:
    return [c.Symbol for c in crypto_coarse if c.Volume >= 100 and c.VolumeInUsd > 10000]
```

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the **AddCrypto** method, call the **RemoveSecurity** method.

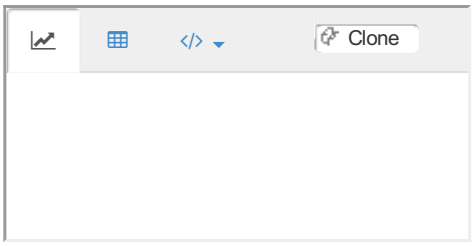
```
PY

self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Example Applications

The Coinbase Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation


Clone

CoinAPI

Kraken Crypto Price Data

Introduction

The Kraken Crypto Price Data by CoinAPI provides Cryptocurrency price and volume data points. The data covers 384 Cryptocurrency pairs, starts in October 2013, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on the Cryptocurrency markets/exchanges supported by QuantConnect.

For more information about the Kraken Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Kraken Crypto Price dataset:

```py
from QuantConnect.DataSource import *
from QuantConnect.Data.UniverseSelection import *

# Kraken accepts both Cash and Margin type account.
self.SetBrokerageModel(BrokerageName.Kraken, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin)

self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.Kraken).Symbol

self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.Kraken, self.UniverseSettings,
self.UniverseSelectionFilter))
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 2013
Asset Coverage	384 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC

Supported Assets

The following table shows the available Cryptocurrency pairs:

Cryptocurrency Pairs					
1INCHEUR	1INCHUSD	AAVEAUD	AAVEBTC	AAVEETH	AAVEEUR
AAVEGBP	AAVEUSD	ADAAUD	ADABTC	ADAETH	ADAEUR
ADAGBP	ADAUSD	ADAUSDT	ALGOBTC	ALGOETH	ALGOEUR
ALGOGBP	ALGOUSD	ANKRBTC	ANKREUR	ANKRGBP	ANKRUSD
ANTBTC	ANTETH	ANTEUR	ANTUSD	ATOMAUD	ATOMBTC
ATOMETH	ATOMEUR	ATOMGBP	ATOMUSD	AUDJPY	AUDUSD
AXSEUR	AXSUSD	BADGEREUR	BADGERUSD	BALBTC	BALETH
BALEUR	BALUSD	BANDEUR	BANDUSD	BATBTC	BATETH
BATEUR	BATUSD	BCHAUD	BCHBTC	BCHETH	BCHEUR
BCHGBP	BCHJPY	BCHUSD	BCHUSDT	BNTBTC	BNTEUR
BNTGBP	BNTUSD	BTCAUD	BTCCAD	BTCCHF	BTCDAI
BTCEUR	BTCGBP	BTCJPY	BTCUSD	BTCUSDC	BTCUSDT
CHZEUR	CHZUSD	COMPBTC	COMPETH	COMPEUR	COMPUSD
CQTEUR	CQTUSD	CRVBTC	CRVETH	CRVEUR	CRVUSD
CTSIEUR	CTSIUSD	DAIEUR	DAIUSD	DAIUSDT	DASHBTC
DASHEUR	DASHUSD	DOTAUD	DOTBTC	DOTETH	DOTEUR
DOTGBP	DOTUSD	DOTUSDT	ENJBTC	ENJEUR	ENJGBP
ENJUSD	EOSBTC	EOSETH	EOSEUR	EOSUSD	EOSUSDT
ETCBTC	ETCETH	ETCEUR	ETCUSD	ETH2.SETH	ETHAUD
ETHBTC	ETHCAD	ETHCHF	ETHDAI	ETHEUR	ETHGBP
ETHJPY	ETHUSD	ETHUSDC	ETHUSDT	EURAUD	EURCAD
EURCHF	EURGBP	EURJPY	EURUSD	EWTBTC	EWTEUR
EWTGBP	EWTUSD	FILAUD	FILBTC	FILETH	FILEUR
FILGBP	FILUSD	FLOWBTC	FLOWETH	FLOWEUR	FLOWGBP

Cryptocurrency Pairs					
FLOWUSD	GBPUSD	GHSTBTC	GHSTEUR	GHSTGBP	GHSTUSD
GNOBTC	GNOETH	GNOEUR	GNOUSD	GRTAUD	GRTBTC
GRTETH	GRTEUR	GRTGBP	GRTUSD	ICXBTC	ICXETH
ICXEUR	ICXUSD	INJEUR	INJUSD	KAREUR	KARUSD
KAVABTC	KAVAETH	KAVAEUR	KAVAUSD	KEEPBTC	KEEPETH
KEEPEUR	KEEPUSD	KNCBTC	KNCETH	KNCEUR	KNCUSD
KSMAUD	KSMBTC	KSMDOT	KSMETH	KSMEUR	KSMGBP
KSMUSD	LINKAUD	LINKBTC	LINKETH	LINKEUR	LINKGBP
LINKUSD	LINKUSDT	LPTBTC	LPTEUR	LPTGBP	LPTUSD
LRCEUR	LRCUSD	LSKBTC	LSKETH	LSKEUR	LSKUSD
LTCAUD	LTCBTC	LTCETH	LTCEUR	LTCGBP	LTCJPY
LTCUSD	LTCUSDT	MANABTC	MANAETH	MANAEUR	MANAUSD
MATICBTC	MATICEUR	MATICGBP	MATICUSD	MINABTC	MINAEUR
MINAGBP	MINAUSD	MIREUR	MIRUSD	MKRBTC	MKREUR
MKRGBP	MKRUSD	MLNBTC	MLNETH	MLNEUR	MLNUSD
MOVREUR	MOVRUSD	NANOBTC	NANOETH	NANOEUR	NANOUSD
OCEANBTC	OCEANEUR	OCEANGBP	OCEANUSD	OGNEUR	OGNUSD
OMGBTC	OMGETH	OMGEUR	OMGUSD	OXTBTC	OXTETH
OXTEUR	OXTUSD	PAXGBTC	PAXGETH	PAXGEUR	PAXGUSD
PERPEUR	PERPUSD	QTUMBTC	QTUMETH	QTUMEUR	QTUMUSD
RARIBTC	RARIEUR	RARIGBP	RARIUSD	RENBTC	RENEUR
RENGBP	RENUSD	REPBTC	REPETH	REPEUR	REPUSD
REPV2BTC	REPV2ETH	REPV2EUR	REPV2USD	SANDBTC	SANDEUR
SANDGBP	SANDUSD	SCBTC	SCETH	SCEUR	SCUSD
SNXAUD	SNXBTC	SNXETH	SNXEUR	SNXGBP	SNXUSD
SOLBTC	SOLEUR	SOLGBP	SOLUSD	SRMBTC	SRMEUR

Cryptocurrency Pairs					
SRMGBP	SRMUSD	STORJBTC	STORJETH	STORJEUR	STORJUSD
SUSHIBTC	SUSHIEUR	SUSHIGBP	SUSHIUSD	TBTCBTC	TBTCETH
TBTCEUR	TBTCUSD	TRXBTC	TRXETH	TRXEUR	TRXUSD
UNIBTC	UNIETH	UNIEUR	UNIUSD	USDCAD	USDCAUD
USDCEUR	USDCGBP	USDCHF	USDCUSD	USDCUSDT	USDJPY
USDTAUD	USDTCAD	USDTCHF	USDTEUR	USDTGBP	USDTJPY
USDTUSD	WAVESBTC	WAVESETH	WAVESEUR	WAVESUSD	WBTCBTC
WBTCEUR	WBTCUSD	XDGBTC	XDGEUR	XDGUSD	XDGUSDT
XLMAUD	XLMBTC	XLMEUR	XLMGBP	XLMUSD	XMRBTC
XMREUR	XMRUSD	XRPAUD	XRPBTC	XRPCAD	XRPETH
XRPEUR	XRPGBP	XRPJPY	XRPUSD	XRPUSDT	XTZAUD
XTZBTC	XTZETH	XTZEUR	XTZGBP	XTZUSD	YFIAUD
YFIBTC	YFIETH	YFIEUR	YFIGBP	YFIUSD	ZECBTC
ZECEUR	ZECUSD	ZRXBTC	ZRXEUR	ZRXGBP	ZRXUSD

Data Point Attributes

The Kraken Crypto Price dataset provides **TradeBar** , **QuoteBar** , **Tick** , and **CryptoCoarseFundamental** objects.

TradeBar Attributes

TradeBar objects have the following attributes:

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

CryptoCoarseFundamental Attributes

CryptoCoarseFundamental objects have the following attributes:

Requesting Data

To add Kraken Crypto Price data to your algorithm, call the **AddCrypto** method. Save a reference to the Crypto **Symbol** so you can access the data later in your algorithm.

```
class CoinAPIDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 6, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        # Kraken accepts both Cash and Margin type account.
        self.SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin)

        self.symbol = self.AddCrypto("BTCUSD", Resolution.Minute, Market.Kraken).Symbol
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Kraken Crypto Price data, index the **Bars** , **QuoteBars** , or **Ticks** properties of the current **Slice** with the Crypto **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        trade_bar = slice.Bars[self.symbol]
        self.Log(f"{self.symbol} close at {slice.Time}: {trade_bar.Close}")

    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
def OnData(self, slice: Slice) -> None:
    for symbol, trade_bar in slice.Bars.items():
        self.Log(f"{symbol} close at {slice.Time}: {trade_bar.Close}")

    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Kraken Crypto Price data, call the **History** method with the Crypto **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Daily)

# TradeBar objects
history_trade_bars = self.History[TradeBar](self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Kraken Crypto pairs, call the **AddUniverse** method with a **CryptoCoarseFundamentalUniverse** object. A Crypto coarse universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
from QuantConnect.Data.UniverseSelection import *

def Initialize(self) -> None:
    self.SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin)
    self.AddUniverse(CryptoCoarseFundamentalUniverse(Market.Kraken, self.UniverseSettings,
self.UniverseSelectionFilter))

def UniverseSelectionFilter(self, crypto_coarse: List[CryptoCoarseFundamental]) -> List[Symbol]:
    return [c.Symbol for c in crypto_coarse if c.Volume >= 100 and c.VolumeInUsd > 10000]
```

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the **AddCrypto** method, call the **RemoveSecurity** method.

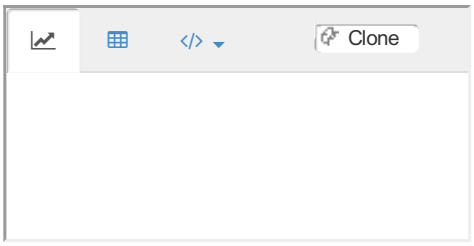
```
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Example Applications

The Kraken Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation


Clone

Datasets

OANDA

OANDA was co-founded by Dr. Stumm, a computer scientist and Dr. Olsen, an economist, in 1997. The company was born out of the belief that the Internet and technology would open up the markets for both currency data and trading. OANDA uses innovative computer and financial technology to provide Internet-based forex trading and currency information services to everyone, from individuals to large corporations, from portfolio managers to financial institutions. OANDA is a market maker and a trusted source for currency data. It has access to one of the world"s largest historical, high-frequency, filtered currency databases.

CFD Data

FOREX Data

OANDA

CFD Data

Introduction

The CFD Data by OANDA serves 269 contracts for differences (CFD). The data starts as early as May 2002 and is delivered on any frequency from tick to daily. This dataset is created by QuantConnect processing raw tick data from OANDA.

For more information about the CFD Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

OANDA was co-founded by Dr. Stumm, a computer scientist and Dr. Olsen, an economist, in 1997. The company was born out of the belief that the Internet and technology would open up the markets for both currency data and trading. OANDA uses innovative computer and financial technology to provide Internet-based forex trading and currency information services to everyone, from individuals to large corporations, from portfolio managers to financial institutions. OANDA is a market maker and a trusted source for currency data. It has access to one of the world's largest historical, high-frequency, filtered currency databases.

Getting Started

The following snippet demonstrates how to request data from the CFD dataset:

```
PY
from QuantConnect.DataSource import *
self.symbol = self.AddCfd("XAUUSD", Resolution.Daily).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	Mixed, earliest starts May 2002
Asset Coverage	269 Contracts
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	Mixed, in which the contract is listed*

* E.g.: DE30EUR tracks DAX30 Index, which is listed in Europe/Berlin timezone.

Supported Assets

The following table shows the available contracts:

Contracts					
AU200AUD	BCOUSD	CH20AUD	CH20CAD	CH20CHF	CH20EUR
CH20GBP	CH20HKD	CH20JPY	CH20SGD	CH20USD	CNHUSD
CORNAUD	CORNCAD	CORNCHF	CORNEUR	CORNGBP	CORNHKD
CORNJPY	CORNSGD	CORNUSD	DE10YBAUD	DE10YBCAD	DE10YBCHF
DE10YBEUR	DE10YBGBP	DE10YBHKD	DE10YBJPY	DE10YBSGD	DE10YBUSD
DE30AUD	DE30CAD	DE30CHF	DE30EUR	DE30GBP	DE30HKD
DE30JPY	DE30SGD	DE30USD	EU50AUD	EU50CAD	EU50CHF
EU50EUR	EU50GBP	EU50HKD	EU50JPY	EU50SGD	EU50USD
FR40AUD	FR40CAD	FR40CHF	FR40EUR	FR40GBP	FR40HKD
FR40JPY	FR40SGD	FR40USD	HK33AUD	HK33CAD	HK33CHF
HK33EUR	HK33GBP	HK33HKD	HK33JPY	HK33SGD	HK33USD
JP225AUD	JP225CAD	JP225CHF	JP225EUR	JP225GBP	JP225HKD
JP225JPY	JP225SGD	JP225USD	NAS100AUD	NAS100CAD	NAS100CHF
NAS100EUR	NAS100GBP	NAS100HKD	NAS100JPY	NAS100SGD	NAS100USD
NATGASAUD	NATGASCAD	NATGASCHF	NATGASEUR	NATGASGBP	NATGASHKD
NATGASJPY	NATGASSGD	NATGASUSD	NL25AUD	NL25CAD	NL25CHF
NL25EUR	NL25GBP	NL25HKD	NL25JPY	NL25SGD	NL25USD
SG30SGD	SOYBNAUD	SOYBNCAD	SOYBNCHF	SOYBNEUR	SOYBNGBP
SOYBNHKD	SOYBNJPY	SOYBNSGD	SOYBNUSD	SPX500AUD	SPX500CAD
SPX500CHF	SPX500EUR	SPX500GBP	SPX500HKD	SPX500JPY	SPX500SGD
SPX500USD	SUGARAUD	SUGARCAD	SUGARCHF	SUGAREUR	SUGARGBP
SUGARHKD	SUGARJPY	SUGARSGD	SUGARUSD	UK100AUD	UK100CAD
UK100CHF	UK100EUR	UK100GBP	UK100HKD	UK100JPY	UK100SGD
UK100USD	UK10YBAUD	UK10YBCAD	UK10YBCHF	UK10YBEUR	UK10YBGBP
UK10YBHKD	UK10YBJPY	UK10YBSGD	UK10YBUSD	US2000AUD	US2000CAD

Contracts					
US2000CHF	US2000EUR	US2000GBP	US2000HKD	US2000JPY	US2000SGD
US2000USD	US30AUD	US30CAD	US30CHF	US30EUR	US30GBP
US30HKD	US30JPY	US30SGD	US30USD	USB02YAUD	USB02YCAD
USB02YCHF	USB02YEUR	USB02YGBP	USB02YHKD	USB02YJPY	USB02YSGD
USB02YUSD	USB05YAUD	USB05YCAD	USB05YCHF	USB05YEUR	USB05YGBP
USB05YHKD	USB05YJPY	USB05YSGD	USB05YUSD	USB10YAUD	USB10YCAD
USB10YCHF	USB10YEUR	USB10YGBP	USB10YHKD	USB10YJPY	USB10YSGD
USB10YUSD	USB30YAUD	USB30YCAD	USB30YCHF	USB30YEUR	USB30YGBP
USB30YHKD	USB30YJPY	USB30YSGD	USB30YUSD	USDCNH	WHEATAUD
WHEATCAD	WHEATCHF	WHEATEUR	WHEATGBP	WHEATHKD	WHEATJPY
WHEATSGD	WHEATUSD	WTICOAUD	WTICOCAD	WTICOCHF	WTICOEUR
WTICOGBP	WTICOHKD	WTICOJPY	WTICOSGD	WTICOUSD	XAGAUD
XAGCAD	XAGCHF	XAGEUR	XAGGBP	XAGHKD	XAGJPY
XAGNZD	XAGSGD	XAGUSD	XAUAUD	XAUCAD	XAUCHF
XAUEUR	XAUGBP	XAUHKD	XAUJPY	XAUNZD	XAUSGD
XAUUSD	XAUXAG	XCUAUD	XCUCAD	XCUCHF	XCUEUR
XCUGBP	XCUHKD	XCUJPY	XCUSGD	XCUUSD	XPDAUD
XPDCAD	XPDCHF	XPDEUR	XPDGBP	XPDHKD	XPDJPY
XPDSGD	XPDUSD	XPTAUD	XPTCAD	XPTCHF	XPTEUR
XPTGBP	XPTHKD	XPTJPY	XPTSGD	XPTUSD	

Data Point Attributes

The CFD dataset provides **QuoteBar** and **Tick** objects.

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

Requesting Data

To add CFD data to your algorithm, call the **AddCfd** method. Save a reference to the CFD **Symbol** so you can access the data later in your algorithm.

```
PY
class CfdAlgorithm (QCAlgorithm):
    def Initialize(self) -> None:
        self.SetAccountCurrency('EUR');

        self.SetStartDate(2019, 2, 20)
        self.SetEndDate(2019, 2, 21)
        self.SetCash('EUR', 100000)

        self.symbol = self.AddCfd('DE30EUR').Symbol

        self.SetBenchmark(self.symbol)
```

For more information about creating CFD subscriptions, see Requesting Data .

Accessing Data

To get the current CFD data, index the **QuoteBars** or **Ticks** properties of the current **Slice** with the CFD **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing CFD data, see Handling Data .

Historical Data

To get historical CFD data, call the **History** method with the CFD **Symbol** . If there is no data in the period you request, the history result is empty.

```py
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a CFD subscription, call the **RemoveSecurity** method.

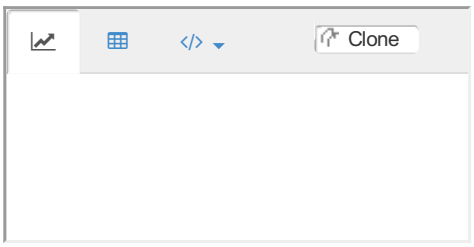
```py
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings.

Example Applications

The CFD price data enables you to trade CFD assets in your algorithm. Examples include the following strategies:

- Exploring the daily worldwide news cycles with CFDs that track international indices.
- Trading price movements of commodities with no delivery of physical goods. For example, pairs trading between gold and silver, corn and wheat, brent and crude oil, etc.



OANDA

FOREX Data

Introduction

The FOREX Data by OANDA serves 185 foreign exchange (FOREX) pairs, starts on various dates from January 2007, and is delivered on any frequency from tick to daily. This dataset is created by QuantConnect processing raw tick data from OANDA.

For more information about the FOREX Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

OANDA was co-founded by Dr. Stumm, a computer scientist and Dr. Olsen, an economist, in 1997. The company was born out of the belief that the Internet and technology would open up the markets for both currency data and trading. OANDA uses innovative computer and financial technology to provide Internet-based forex trading and currency information services to everyone, from individuals to large corporations, from portfolio managers to financial institutions. OANDA is a market maker and a trusted source for currency data. It has access to one of the world's largest historical, high-frequency, filtered currency databases.

Getting Started

The following snippet demonstrates how to request data from the FOREX dataset:

```
PY
from QuantConnect.DataSource import *
self.symbol = self.AddForex("EURUSD", Resolution.Daily, Market.Oanda).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2007
Asset Coverage	71 Currency pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	UTC

Supported Assets

The following table shows the available Forex pairs:

Forex Pairs					
AUDCAD	AUDCHF	AUDHKD	AUDJPY	AUDNZD	AUDSGD
AUDUSD	CADCHF	CADHKD	CADJPY	CADSGD	CHFHKD
CHFJPY	CHFZAR	EURAUD	EURCAD	EURCHF	EURCZK
EURDKK	EURGBP	EURHKD	EURHUF	EURJPY	EURNOK
EURNZD	EURPLN	EURSEK	EURSGD	EURTRY	EURUSD
EURZAR	GBPAUD	GBPCAD	GBPCHF	GBPHKD	GBPJPY
GBPNZD	GBPPLN	GBPSGD	GBPUSD	GBPZAR	HKDJPY
NZDCAD	NZDCHF	NZDHKD	NZDJPY	NZDSGD	NZDUSD
SGDCHF	SGDHKD	SGDJPY	TRYJPY	USDCAD	USDCHF
USDCNH	USDCZK	USDDKK	USDHKD	USDHUF	USDINR
USDJPY	USDMXN	USDNOK	USDPLN	USDSAR	USDSEK
USDSGD	USDTHB	USDTRY	USDZAR	ZARJPY	

Data Point Attributes

The FOREX dataset provides **QuoteBar** and **Tick** objects.

QuoteBar Attributes

QuoteBar objects have the following attributes:

Tick Attributes

Tick objects have the following attributes:

Requesting Data

To add FOREX data to your algorithm, call the **AddForex** method. Save a reference to the Forex **Symbol** so you can access the data later in your algorithm.

```
class ForexAlgorithm (QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 2, 20)
        self.SetEndDate(2019, 2, 21)
        self.SetCash(100000)

        self.symbol = self.AddForex('EURUSD', Resolution.Minute, Market.Oanda).Symbol

        self.SetBenchmark(self.symbol)
```

PY

For more information about creating Forex subscriptions, see Requesting Data .

Accessing Data

To get the current Forex data, index the **QuoteBars** or **Ticks** properties of the current **Slice** with the Forex **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your security at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.QuoteBars:
        quote_bar = slice.QuoteBars[self.symbol]
        self.Log(f"{self.symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    if self.symbol in slice.Ticks:
        ticks = slice.Ticks[self.symbol]
        for tick in ticks:
            self.Log(f"{self.symbol} price at {slice.Time}: {tick.Price}")
```

You can also iterate through all of the data objects in the current **Slice** .

```
PY
def OnData(self, slice: Slice) -> None:
    for symbol, quote_bar in slice.QuoteBars.items():
        self.Log(f"{symbol} bid at {slice.Time}: {quote_bar.Bid.Close}")

    for symbol, ticks in slice.Ticks.items():
        for tick in ticks:
            self.Log(f"{symbol} price at {slice.Time}: {tick.Price}")
```

For more information about accessing Forex data, see Handling Data .

Historical Data

To get historical Forex data, call the **History** method with the Forex **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.symbol, 100, Resolution.Minute)

# QuoteBar objects
history_quote_bars = self.History[QuoteBar](self.symbol, 100, Resolution.Minute)

# Tick objects
history_ticks = self.History[Tick](self.symbol, timedelta(seconds=10), Resolution.Tick)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a Forex pair subscription, call the **RemoveSecurity** method.

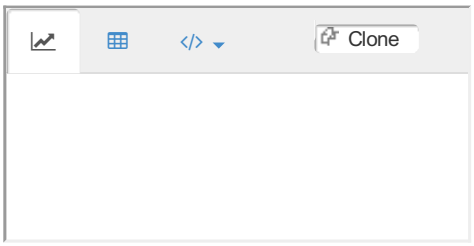
```
PY
self.RemoveSecurity(self.symbol)
```

The **RemoveSecurity** method cancels your open orders for the security and liquidates your holdings.

Example Applications

The FOREX price data enables you to trade currency pairs in the global mark. Examples include the following strategies:

- Exploring the impact that daily worldwide news cycles has on international currencies
- Carry Trade: borrow from a lower interest currency pair to fund the purchase of a currency pair with a higher interest rate



Datasets

Benzinga

Benzinga was founded by Jason Raznick in 2010 with goal of connecting the world with news, data, and education that makes the path to financial prosperity easier for everyone, everyday. Benzinga provides access to real-time news for individual investors.

Benzinga News Feed

Benzinga

Benzinga News Feed

Introduction

The Benzinga News Feed dataset by Benzinga tracks US Equity news releases. The data covers about 1,250 articles per day across 8,000 Equities, starts in January 2016, and is delivered on a second frequency. This dataset is created by structuring the content produced by Benzinga's editorial team.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Benzinga News Feed dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Benzinga was founded by Jason Raznick in 2010 with goal of connecting the world with news, data, and education that makes the path to financial prosperity easier for everyone, everyday. Benzinga provides access to real-time news for individual investors.

Getting Started

The following snippet demonstrates how to request data from the Benzinga News Feed dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(BenzingaNews, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	September 2017
Asset Coverage	8,000 Equities
Data Density	Sparse
Resolution	Second (1,250 Articles/Day)
Timezone	New York

Data Point Attributes

The Benzinga News Feed dataset provides **BenzingaNews** objects, which have the following attributes:

Requesting Data

To add Benzinga News Feed data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY

class BenzingaNewsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
        self.dataset_symbol = self.AddData(BenzingaNews, self.symbol).Symbol
```

Accessing Data

To get the current Benzinga News Feed data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        article = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} title at {slice.Time}: {article.Title}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY

def OnData(self, slice: Slice) -> None:
    for dataset_symbol, article in slice.Get(BenzingaNews).items():
        self.Log(f"{dataset_symbol} title at {slice.Time}: {article.Title}")
```

Historical Data

To get historical Benzinga News Feed data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY

# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[BenzingaNews](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

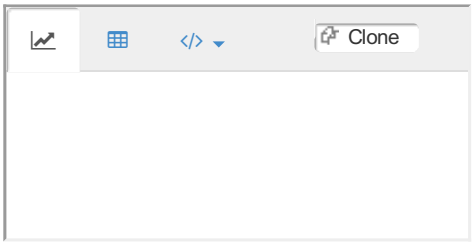
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Benzinga News Feed data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Benzinga News Feed enables you to accurately design strategies harnessing real-time news releases. Examples include the following strategies:

- Creating a dictionary of sentiment scores for various words and assigning a sentiment score to the content of each news release
- Calculating the sentiment of news releases with Natural Language Processing (NLP)
- Trading securities when their news releases that Benzinga tags with current buzzwords



Datasets

Blockchain

Blockchain is a website that publishes data related to Bitcoin. It has been online since 2011 and publishes the Bitcoin Metadata history back to 2009.

Bitcoin Metadata

Blockchain

Bitcoin Metadata

Introduction

The Bitcoin Metadata dataset by Blockchain provides 23 fundamental metadata of Bitcoin directly fetched from the Bitcoin blockchain. The data starts in January 2009 and delivered on a daily frequency. This dataset contains mining statistics like hash rate and miner revenue; transaction metadata like transaction per block, transaction fee, and number of addresses; and blockchain metadata like blockchain size and block size.

For more information about the Bitcoin Metadata dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Blockchain is a website that publishes data related to Bitcoin. It has been online since 2011 and publishes the Bitcoin Metadata history back to 2009.

Getting Started

The following snippet demonstrates how to request data from the Bitcoin Metadata dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddCrypto("BTCUSD", Resolution.Daily, Market.Bitfinex).Symbol
self.dataset_symbol = self.AddData(BitcoinMetadata, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2009
Coverage	Bitcoin blockchain
Data Density	Regular
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Bitcoin Metadata dataset provides **BitcoinMetadata** objects, which have the following attributes:

Requesting Data

To add Bitcoin Metadata data to your algorithm, call the **AddData** method with the BTCUSD **Symbol** . Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY

class BlockchainBitcoinMetadataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddCrypto("BTCUSD", Resolution.Daily, Market.Bitfinex).Symbol
        self.dataset_symbol = self.AddData(BitcoinMetadata, self.symbol).Symbol
```

Accessing Data

To get the current Bitcoin Metadata data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} miner revenue at {slice.Time}: {data_point.MinersRevenue}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY

def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(BlockchainBitcoinData).items():
        self.Log(f"{dataset_symbol} miner revenue at {slice.Time}: {data_point.MinersRevenue}")
```

Historical Data

To get historical Bitcoin Metadata data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY

# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[BlockchainBitcoinData](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

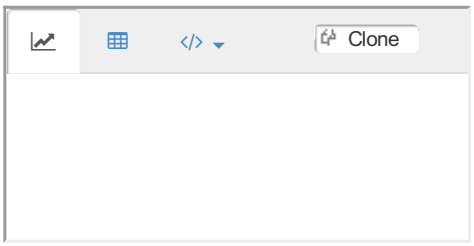
To remove a subscription, call the **RemoveSecurity** method.

```
PY

self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The Bitcoin Metadata dataset enables you to incorporate metadata from the Bitcoin blockchain into your strategies. Examples include the following strategies:

- Comparing mining and transaction statistics to provide insight on the supply-demand relationship of the Bitcoin blockchain service.
- Measuring the activity and popularity of the Bitcoin blockchain to predict the price movements of the Cryptocurrency.



Datasets

Brain

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Brain Language Metrics on Company Filings

Brain ML Stock Ranking

Brain Sentiment Indicator

Brain

Brain Language Metrics on Company Filings

Introduction

The Brain Language Metrics on Company Filings dataset provides the results of an NLP system that monitors several language metrics on 10-K and 10-Q company reports for US Equities. The data covers 5,000 US Equities, starts in January 2010, and is delivered on a daily frequency. The dataset is made of two parts; the first one includes the language metrics of the most recent 10-K or 10-Q report for each firm, namely:

1. Financial sentiment
2. Percentage of words belonging to financial domain classified by language types (e.g. “litigious” or “constraining” language)
3. Readability score
4. Lexical metrics such as lexical density and richness
5. Text statistics such as the report length and the average sentence length

The second part includes the differences between the two most recent 10-Ks or 10-Qs reports of the same period for each company, namely:

1. Difference of the various language metrics (e.g. delta sentiment, delta readability score, delta percentage of a specific language type etc.)
2. Similarity metrics between documents, also with respect to a specific language type (for example similarity with respect to “litigious” language or “uncertainty” language)

The analysis is available for the whole report and for specific sections of the report (e.g. Risk Factors and MD&A).

For more information, refer to Brain's summary paper .

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Brain Language Metrics on Company Filings dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain Language Metrics on Company Filings dataset:

```
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_10k_symbol = self.AddData(BrainCompanyFilingLanguageMetrics10K , self.symbol).Symbol
self.dataset_all_symbol = self.AddData(BrainCompanyFilingLanguageMetricsAll, self.symbol).Symbol

self.AddUniverse(BrainCompanyFilingLanguageMetricsUniverse10K,
"BrainCompanyFilingLanguageMetricsUniverse10K", Resolution.Daily, self.UniverseSelection)
self.AddUniverse(BrainCompanyFilingLanguageMetricsUniverseAll,
"BrainCompanyFilingLanguageMetricsUniverseAll", Resolution.Daily, self.UniverseSelection)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2010
Asset Coverage	5,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Brain Language Metrics on Company Filings dataset provides **BrainCompanyFilingLanguageMetrics** and **BrainCompanyFilingLanguageMetricsUniverse** objects.

BrainCompanyFilingLanguageMetrics Attributes

BrainCompanyFilingLanguageMetrics objects have the following attributes:

BrainCompanyFilingLanguageMetricsUniverse Attributes

BrainCompanyFilingLanguageMetricsUniverse objects have the following attributes:

Requesting Data

To add Brain Language Metrics on Company Filings data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class BrainCompanyFilingNLPDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2010, 1, 1)
        self.SetEndDate(2021, 7, 8)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_10k_symbol = self.AddData(BrainCompanyFilingLanguageMetrics10K, self.symbol).Symbol
        self.dataset_all_symbol = self.AddData(BrainCompanyFilingLanguageMetricsAll, self.symbol).Symbol
```

Accessing Data

To get the current Brain Language Metrics on Company Filings data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_10k_symbol):
        data_point = slice[self.dataset_10k_symbol]
        self.Log(f"{self.dataset_10k_symbol} report sentiment at {slice.Time}:
{data_point.ReportSentiment.Sentiment}")

    if slice.ContainsKey(self.dataset_all_symbol):
        data_point = slice[self.dataset_all_symbol]
        self.Log(f"{self.dataset_all_symbol} report sentiment at {slice.Time}:
{data_point.ReportSentiment.Sentiment}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY

def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(BrainCompanyFilingLanguageMetrics10K).items():
        self.Log(f"{dataset_symbol} report sentiment at {slice.Time}:
{data_point.ReportSentiment.Sentiment}")

    for dataset_symbol, data_point in slice.Get(BrainCompanyFilingLanguageMetricsAll).items():
        self.Log(f"{dataset_symbol} report sentiment at {slice.Time}:
{data_point.ReportSentiment.Sentiment}")
```

Historical Data

To get historical Brain Language Metrics on Company Filings data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY

# DataFrames
ten_k_history_df = self.History(self.dataset_10k_symbol, 100, Resolution.Daily)
all_history_df = self.History(self.dataset_all_symbol, 100, Resolution.Daily)
history_df = self.History([self.dataset_10k_symbol, self.dataset_all_symbol], 100, Resolution.Daily)

# Dataset objects
ten_k_history_bars = self.History[BrainCompanyFilingLanguageMetrics10K](self.dataset_10k_symbol, 100,
Resolution.Daily)
all_history_bars = self.History[BrainCompanyFilingLanguageMetricsAll](self.dataset_all_symbol, 100,
Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Brain Language Metrics on Company Filings data, call the **AddUniverse** method with the **BrainCompanyFilingLanguageMetricsUniverseAll** class or the **BrainCompanyFilingLanguageMetricsUniverse10K** class and a selection function.

```
PY
def Initialize(self) -> None:
    self.AddUniverse(BrainCompanyFilingLanguageMetricsUniverseAll,
"BrainCompanyFilingLanguageMetricsUniverseAll", Resolution.Daily, self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[BrainCompanyFilingLanguageMetricsUniverseAll]) ->
List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.ReportSentiment.Sentiment > 0 \
                and d.ManagementDiscussionAnalyasisOfFinancialConditionAndResultsOfOperations.Sentiment >
0]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
PY
self.RemoveSecurity(self.dataset_10k_symbol)
self.RemoveSecurity(self.dataset_all_symbol)
```

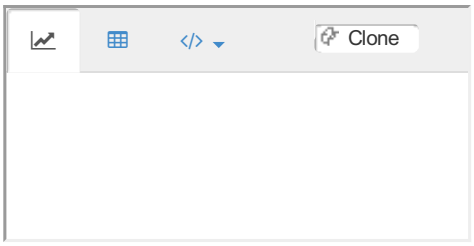
If you subscribe to Brain Language Metrics on Company Filings data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Brain Language Metrics on Company Filings dataset enables you to test strategies using language metrics and their differences gathered from 10K and 10Q reports. Examples include the following strategies:

- Using the similarity among reports to determine position sizing of securities. Some examples are discussed in Lazy Prices, Cohen et al. 2018 and The Positive Similarity of Company Filings and the Cross-section of Stock Returns, M. Padyšák 2020 .
- Using the sentiment of the latest report to determine the portfolio allocation to give to each security in the universe.
- Using levels of uncertainty, readability, or litigious language in the report to determine position sizing of securities.

Disclaimer: The dataset is provided by the data provider for informational purposes only and does not constitute an offer to sell, a solicitation to buy, or a recommendation or endorsement for any security or strategy, nor do they constitute an offer to provide investment advisory or other services by the data provider.


Clone

Brain

Brain ML Stock Ranking

Introduction

The Brain ML Stock Ranking dataset by Brain generates a daily ranking for US Equities based on their predicted ranking of future returns relative to the universe median across four-time horizons: next 2, 3, 5, 10, and 21 days (one trading month). The data covers 1,000 US Equities (universe updated yearly by including the largest 1,000 US companies of the previous year), starts in January 2010, and is delivered on a daily frequency. This dataset is created by a voting scheme of machine learning classifiers that non-linearly combine a variety of features with a series of techniques aimed at mitigating the well-known overfitting problem for financial data with a low signal-to-noise ratio. Examples of features are time-varying stock-specific features like price and volume-related metrics or fundamentals; time-fixed stock-specific features like the sector and other database information; market regime features such as volatility and other financial stress indicators; calendar features representing possible anomalies, for example, the month of the year.

More precisely the ML Stock Ranking score is related to the confidence of a Machine Learning classifier in predicting top or bottom quintile returns for the next N trading days (e.g. next 21 trading days) for a stock with the respect to the median of the universe and ranges from -1 to +1.

A negative score means that the system is more confident that the stock belongs to the lower returns quintile, a positive score means that the system is more confident that the stock belongs to the higher returns quintile. It is important to note that the score has a meaning only if used to compare different stocks to perform a ranking.

Typical use is to download the score for a large stock universe for a given day, e.g. 500 stocks or the full universe of 1000 stocks, order the stocks by mlAlpha score and go long the top K stocks, or build a long-short strategy going long the top K and short the bottom K stocks.

For more information, refer to Brain's summary paper .

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Brain ML Stock Ranking dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain ML Stock Ranking dataset:

```
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(BrainStockRanking2Day, self.symbol).Symbol

self.AddUniverse(BrainStockRankingUniverse, "BrainStockRankingUniverse", Resolution.Daily,
self.UniverseSelectionMethod)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2010
Asset Coverage	1,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Brain ML Stock Ranking dataset provides **BrainStockRankingBase** and **BrainStockRankingUniverse** objects.

BrainStockRankingBase Attributes

BrainStockRankingBase objects have the following attributes:

BrainStockRankingUniverse Attributes

BrainStockRankingUniverse objects have the following attributes:

Requesting Data

To add Brain ML Stock Ranking data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class BrainMLRankingDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 7, 8)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.two_day_symbol = self.AddData(BrainStockRanking2Day, self.symbol).Symbol
        self.three_day_symbol = self.AddData(BrainStockRanking3Day, self.symbol).Symbol
        self.five_day_symbol = self.AddData(BrainStockRanking5Day, self.symbol).Symbol
        self.ten_day_symbol = self.AddData(BrainStockRanking10Day, self.symbol).Symbol
        self.month_symbol = self.AddData(BrainStockRanking21Day, self.symbol).Symbol
```

Accessing Data

To get the current Brain ML Stock Ranking data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.two_day_symbol):
        data_point = slice[self.two_day_symbol]
        self.Log(f"{self.two_day_symbol} rank at {slice.Time}: {data_point.Rank}")

    if slice.ContainsKey(self.three_day_symbol):
        data_point = slice[self.three_day_symbol]
        self.Log(f"{self.three_day_symbol} rank at {slice.Time}: {data_point.Rank}")

    if slice.ContainsKey(self.five_day_symbol):
        data_point = slice[self.five_day_symbol]
        self.Log(f"{self.five_day_symbol} rank at {slice.Time}: {data_point.Rank}")

    if slice.ContainsKey(self.ten_day_symbol):
        data_point = slice[self.ten_day_symbol]
        self.Log(f"{self.ten_day_symbol} rank at {slice.Time}: {data_point.Rank}")

    if slice.ContainsKey(self.month_symbol):
        data_point = slice[self.month_symbol]
        self.Log(f"{self.month_symbol} rank at {slice.Time}: {data_point.Rank}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(BrainStockRanking2Day).items():
        self.Log(f"{dataset_symbol} rank at {slice.Time}: {data_point.Rank}")

    for dataset_symbol, data_point in slice.Get(BrainStockRanking3Day).items():
        self.Log(f"{dataset_symbol} rank at {slice.Time}: {data_point.Rank}")

    for dataset_symbol, data_point in slice.Get(BrainStockRanking5Day).items():
        self.Log(f"{dataset_symbol} rank at {slice.Time}: {data_point.Rank}")

    for dataset_symbol, data_point in slice.Get(BrainStockRanking10Day).items():
        self.Log(f"{dataset_symbol} rank at {slice.Time}: {data_point.Rank}")

    for dataset_symbol, data_point in slice.Get(BrainStockRanking21Day).items():
        self.Log(f"{dataset_symbol} rank at {slice.Time}: {data_point.Rank}")
```

Historical Data

To get historical Brain ML Stock Ranking data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrames
two_day_history_df = self.History(self.two_day_symbol, 100, Resolution.Daily)
three_day_history_df = self.History(self.three_day_symbol, 100, Resolution.Daily)
five_day_history_df = self.History(self.five_day_symbol, 100, Resolution.Daily)
ten_day_history_df = self.History(self.ten_day_symbol, 100, Resolution.Daily)
thirty_day_history_df = self.History(self.month_symbol, 100, Resolution.Daily)
history_df = self.History([self.two_day_symbol,
                           self.three_day_symbol,
                           self.five_day_symbol,
                           self.ten_day_symbol,
                           self.month_symbol], 100, Resolution.Daily)

# Dataset objects
two_day_history_bars = self.History[BrainStockRanking2Day](self.two_day_symbol, 100, Resolution.Daily)
three_day_history_bars = self.History[BrainStockRanking3Day](self.three_day_symbol, 100, Resolution.Daily)
five_day_history_bars = self.History[BrainStockRanking5Day](self.five_day_symbol, 100, Resolution.Daily)
ten_day_history_bars = self.History[BrainStockRanking10Day](self.ten_day_symbol, 100, Resolution.Daily)
month_history_bars = self.History[BrainStockRanking21Day](self.month_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Brain ML Stock Ranking data, call the **AddUniverse** method with the **BrainStockRankingUniverse** class and a selection function.

```
def Initialize(self) -> None:
    self.AddUniverse(BrainStockRankingUniverse, "BrainStockRankingUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[BrainStockRankingUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.Rank2Days > 0 \
                and d.Rank3Days > 0 \
                and d.Rank5Days > 0]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.two_day_symbol)
self.RemoveSecurity(self.three_day_symbol)
self.RemoveSecurity(self.five_day_symbol)
self.RemoveSecurity(self.ten_day_symbol)
self.RemoveSecurity(self.month_symbol)
```

If you subscribe to Brain ML Stock Ranking data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

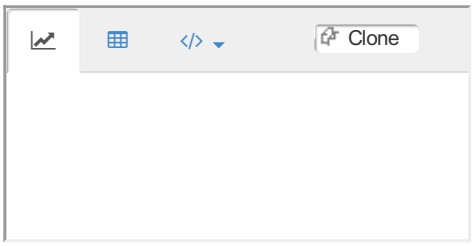
Example Applications

The Brain ML Stock Ranking dataset enables you to test strategies using the machine learning ranking provided by Brain. Examples include the following strategies:

- Constructing a portfolio of securities with each security's weight in the portfolio reflecting its Brain ML Stock Ranking

- Buying stocks with the largest Brain ML Stock Ranking
- Building a market-neutral strategy based on the top N and bottom N stocks in the Brain ML Stock Ranking

Disclaimer: The dataset is provided by the data provider for informational purposes only and do not constitute an offer to sell, a solicitation to buy, or a recommendation or endorsement for any security or strategy, nor do they constitute an offer to provide investment advisory or other services by the data provider.



Brain

Brain Sentiment Indicator

Introduction

The Brain Sentiment Indicator dataset by Brain tracks the public sentiment around US Equities. The data covers 4,500 US Equities, starts in August 2016, and is delivered on a daily frequency. This dataset is created by analyzing financial news using Natural Language Processing techniques while taking into account the similarity and repetition of news on the same topic. The sentiment score assigned to each stock ranges from -1 (most negative) to +1 (most positive). The sentiment score corresponds to the average sentiment for each piece of news. The score is updated daily and is available on two time scales: 7 days and 30 days. For more information, see Brain's summary paper .

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Brain Sentiment Indicator dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain Sentiment Indicator dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_7day_symbol = self.AddData(BrainSentimentIndicator7Day, self.symbol).Symbol
self.dataset_30day_symbol = self.AddData(BrainSentimentIndicator30Day, self.symbol).Symbol

self.AddUniverse(BrainSentimentIndicatorUniverse, "BrainSentimentIndicatorUniverse", Resolution.Daily,
self.UniverseSelectionMethod)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2016
Asset Coverage	4,500 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Brain Sentiment Indicator dataset provides **BrainSentimentIndicatorBase** and **BrainSentimentIndicatorUniverse** objects.

BrainSentimentIndicatorBase Attributes

BrainSentimentIndicatorBase objects have the following attributes:

BrainSentimentIndicatorUniverse Attributes

BrainSentimentIndicatorUniverse objects have the following attributes:

Requesting Data

To add Brain Sentiment Indicator data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class BrainSentimentDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2021, 7, 8)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_7day_symbol = self.AddData(BrainSentimentIndicator7Day, self.symbol).Symbol
        self.dataset_30day_symbol = self.AddData(BrainSentimentIndicator30Day, self.symbol).Symbol
```

Accessing Data

To get the current Brain Sentiment Indicator data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_7day_symbol):
        data_point = slice[self.dataset_7day_symbol]
        self.Log(f"{self.dataset_7day_symbol} sentiment at {slice.Time}: {data_point.Sentiment}")

    if slice.ContainsKey(self.dataset_30day_symbol):
        data_point = slice[self.dataset_30day_symbol]
        self.Log(f"{self.dataset_30day_symbol} sentiment at {slice.Time}: {data_point.Sentiment}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(BrainSentimentIndicator7Day).items():
        self.Log(f"{dataset_symbol} sentiment at {slice.Time}: {data_point.Sentiment}")

    for dataset_symbol, data_point in slice.Get(BrainSentimentIndicator30Day).items():
        self.Log(f"{dataset_symbol} sentiment at {slice.Time}: {data_point.Sentiment}")
```

Historical Data

To get historical Brain Sentiment Indicator data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrames
week_history_df = self.History(self.dataset_7day_symbol, 100, Resolution.Daily)
month_history_df = self.History(self.dataset_30day_symbol, 100, Resolution.Daily)
history_df = self.History([self.dataset_7day_symbol, self.dataset_30day_symbol], 100, Resolution.Daily)

# Dataset objects
week_history_bars = self.History[BrainSentimentIndicator7Day](self.dataset_7day_symbol, 100,
Resolution.Daily)
month_history_bars = self.History[BrainSentimentIndicator30Day](self.dataset_30day_symbol, 100,
Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Brain Sentiment Indicator data, call the **AddUniverse** method with the **BrainSentimentIndicatorUniverse** class and a selection function.

```
PY
def Initialize(self) -> None:
    self.AddUniverse(BrainSentimentIndicatorUniverse, "BrainSentimentIndicatorUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[BrainSentimentIndicatorUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.TotalArticleMentions7Days > 0 \
                and d.Sentiment7Days]
```

The Brain Sentiment Indicator universe runs at 7 AM Eastern Time (ET) in live trading. For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
PY
self.RemoveSecurity(self.dataset_7day_symbol)
self.RemoveSecurity(self.dataset_30day_symbol)
```

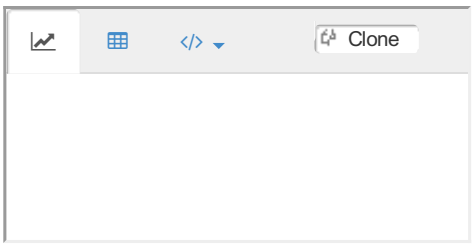
If you subscribe to Brain Sentiment Indicator data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Brain Sentiment Indicator dataset enables you to incorporate sentiment from financial news sources into your strategies. Examples include the following strategies:

- Buying when the public sentiment for a security is increasing
- Short selling when the public sentiment for a security is decreasing
- Scaling the position sizing of securities based on how many times they are mentioned in financial news articles

Disclaimer: The dataset is provided by the data provider for informational purposes only and does not constitute an offer to sell, a solicitation to buy, or a recommendation or endorsement for any security or strategy, nor do they constitute an offer to provide investment advisory or other services by the data provider.



Datasets

CBOE

The Chicago Board Options Exchange (CBOE) is the largest U.S. options exchange with annual trading volume that hovered around 1.27 billion contracts at the end of 2014. CBOE offers Options on over 2,200 companies, 22 Equity indices, and 140 exchange-traded funds (ETFs).

VIX Daily Price

CBOE

VIX Daily Price

Introduction

The VIX Daily Price dataset by CBOE covers 14 US volatility indices. The data starts in January 1990 and is delivered on a daily frequency. The dataset is cached daily from the CBOE website. The volatility index measures the stock market's expectation of volatility on the market index (e.g.: S&P500) using implied volatility from its Options for a fixed time horizon.

For more information about the VIX Daily Price dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Chicago Board Options Exchange (CBOE) is the largest U.S. options exchange with annual trading volume that hovered around 1.27 billion contracts at the end of 2014. CBOE offers Options on over 2,200 companies, 22 Equity indices, and 140 exchange-traded funds (ETFs).

Getting Started

The following snippet demonstrates how to request data from the VIX Daily Price dataset:

```
PY
from QuantConnect.DataSource import *
self.dataset_symbol = self.AddData(CBOE, "VIX", Resolution.Daily).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1990
Asset Coverage	14 US Volatility Indices
Data Density	Regular
Resolution	Daily
Timezone	New York

Supported Indices

The following table shows the volatility indices in the VIX Daily Price dataset:

Ticker	Index	Expiry	Start Date	Data Points
VIX	S&P500	30 Days	Jan 1990	OHLC
VIX9D	S&P500	9 Days	Apr 2011	OHLC
VIX3M	S&P500	3 Months	Sep 2009	OHLC
VIX6M	S&P500	6 Months	Jan 2008	OHLC
VIX1Y	S&P500	1 Year	Jan 2007	OHLC
VXO	S&P100	30 Days	Feb 1993	OHLC
VXN	Nasdaq 100	30 Days	Sep 2009	OHLC
RVX	Russell 2000	30 Days	Sep 2009	OHLC
VVIX	VIX	30 Days	Mar 2006	Close
TYVIX	10-year US Treasury Note	30 Days	Jan 2003	OHLC
VXTLT	20-year US Treasury Bond	30 Days	Jan 2004	Close
VXEEM	MSCI Emerging Markets	30 Days	Mar 2011	OHLC
OVX	United States Oil Fund (USO)	30 Days	Sep 2009	Close
GVZ	SPDR Gold Shares ETF (GLD)	30 Days	Sep 2009	Close

Data Point Attributes

The VIX Daily Price dataset provides **CBOE** objects, which have the following attributes:

Requesting Data

To add VIX Daily Price data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class CboeDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2003, 1, 1)
        self.SetEndDate(2019, 10, 11)
        self.SetCash(100000)

        self.dataset_symbol = self.AddData(CBOE, "VIX", Resolution.Daily).Symbol
```

Accessing Data

To get the current VIX Daily Price data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} close at {slice.Time}: {data_point.Close}")
```

Historical Data

To get historical VIX Daily Price data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[CBOE](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

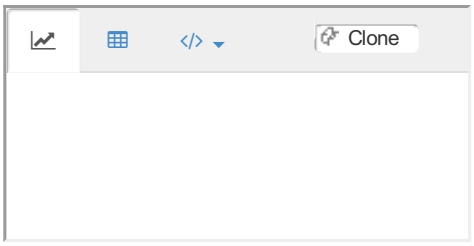
To remove your subscription to VIX Daily Price data, call the **RemoveSecurity** method.

```
PY
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The VIX Daily Price enables you to incorporate popular US volatility indicies in your strategies. Examples include the following strategies:

- Understanding the stock market's level of expected forward-looking volatility, also known as the "fear index". When the VIX starts moving higher, it is telling you that traders are getting nervous. When the VIX starts moving lower, it is telling you that traders are gaining confidence.
- Determining forward-looking volatility by comparing the VIX against volatility indices with other volatility. By comparing the value of the VIX to the value of the VIX3M, you can identify periods when trader sentiment has turned extremely bearish and when it has normalized.


Clone

Datasets

CoinGecko

CoinGecko was founded in 2014 by TM Lee (CEO) and Bobby Ong (COO) with the mission to democratize the access of crypto data and empower users with actionable insights. We also deep dive into the crypto space to deliver valuable insights to our users through our cryptocurrency reports, as well as our publications, newsletter, and more.

Crypto Market Cap

CoinGecko

Crypto Market Cap

Introduction

The Crypto Market Cap dataset by CoinGecko tracks the market cap of cryptocurrencies. The data covers 620 cryptocurrencies that are supported by QuantConnect, starts in 28 April 2013, and is delivered on a daily frequency. This dataset is created by scraping CoinGecko's Market Chart.

For more information about the Crypto Market Cap dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinGecko was founded in 2014 by TM Lee (CEO) and Bobby Ong (COO) with the mission to democratize the access of crypto data and empower users with actionable insights. We also deep dive into the crypto space to deliver valuable insights to our users through our cryptocurrency reports, as well as our publications, newsletter, and more.

Getting Started

The following snippet demonstrates how to request data from the CoinGecko Crypto Market Cap dataset:

```
PY
self.symbol = self.AddData(CoinGecko, "BTC").Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	28 April 2013
Asset Coverage	620 cryptocurrencies
Resolution	Daily
Timezone	UTC

Supported Assets

Supported Cryptocurrency					
1INCH	AAA	AAPL	AAVE	ACA	ACH
ACM	ADA	ADX	AE	AERGO	AGIX

Supported Cryptocurrency					
AGLD	AION	AKRO	ALBT	ALCX	ALEPH
ALGO	ALICE	ALPACA	ALPHA	ALPINE	AMB
AMC	AMP	AMPL	ANC	ANKR	ANT
ANY	APE	API3	APPC	APT	AR
ARDR	ARK	ARPA	ASD	ASM	ASR
AST	ASTR	ATA	ATLAS	ATM	ATOM
AUCTION	AUDIO	AURY	AUTO	AVA	AVAX
AVT	AXS	B21	BADGER	BAKE	BAL
BAND	BAO	BAR	BAT	BB	BCC
BCD	BCH	BCN	BCPT	BDOT	BEAM
BEL	BEST	BETA	BETH	BFT	BICO
BIDR	BIFI	BIT	BITO	BLT	BLZ
BMI	BNB	BNT	BNTX	BNX	BOBA
BOND	BOSON	BOT	BRD	BRL	BRZ
BSV	BSW	BTC	BTCB	BTCST	BTG
BTRST	BTS	BTSE	BTT	BULL	BURGER
BUSD	BVOL	BYND	BZRX	C98	CAKE
CCD	CDT	CEL	CELO	CELR	CFX
CGC	CHAT	CHESS	CHEX	CHR	CHSB
CHZ	CITY	CKB	CLO	CLOAK	CLV
CMT	CND	CNHT	COCOS	COIN	COMP
CONV	COPE	COS	COTI	COVAL	COVER
CQT	CREAM	CRO	CRON	CRV	CTK
CTSI	CTX	CTXC	CUSDT	CVC	CVP
CVX	DAI	DAR	DASH	DATA	DAWN
DCR	DDX	DEGO	DENT	DESO	DEXE

Supported Cryptocurrency					
DF	DFL	DGB	DGD	DIA	DLT
DMG	DNT	DOCK	DODO	DOGE	DORA
DOT	DREP	DUSK	DVF	DYDX	EDEN
EGLD	ELF	EMB	ENG	ENJ	ENS
EOS	EPS	EPX	ERN	ESS	ETC
ETH	ETH2	ETHE	ETP	EURS	EURT
EVX	EWT	EXO	EZ	FARM	FB
FCL	FET	FIDA	FIL	FIO	FIRO
FIS	FLM	FLOW	FLUX	FOR	FORTH
FOX	FRONT	FTM	FTT	FUEL	FUN
FX	FXS	GAL	GALA	GAS	GENE
GFI	GHST	GLD	GLM	GLMR	GLXY
GME	GMT	GMX	GNO	GNT	GO
GODS	GOG	GOT	GRS	GRT	GT
GTC	GTO	GTX	GVT	GYEN	HARD
HBAR	HC	HEGIC	HFT	HGET	HIGH
HIVE	HMT	HNT	HOLY	HOOK	HOT
HT	HUM	HXRO	ICE	ICN	ICP
ICX	ID	IDEX	IDRT	ILV	IMX
INJ	INS	INTER	INV	IOST	IOTX
IQ	IRIS	JASMY	JET	JOE	JST
JUV	KAI	KAN	KAR	KAVA	KDA
KEEP	KEY	KIN	KLAY	KMD	KNC
KP3R	KRL	KSHIB	KSM	LAZIO	LCX
LDO	LEND	LEO	LEVER	LINA	LINK
LIT	LOKA	LOOKS	LOOM	LPT	LQTY

Supported Cryptocurrency					
LRC	LSK	LTC	LTO	LUA	LUN
LUNA	LUNC	LYM	MAGIC	MANA	MAPS
MASK	MATH	MATIC	MBL	MBOX	MBS
MC	MCB	MCO	MCO2	MDA	MDT
MDX	MEDIA	MER	MFT	MIM	MINA
MIR	MITH	MKR	MLN	MNGO	MOB
MOD	MOVR	MPL	MSOL	MSTR	MTA
MTH	MTL	MULTI	MUSD	NAS	NAV
NBS	NCASH	NCT	NEAR	NEBL	NEO
NEXO	NKN	NMR	NPXS	NU	NULS
NXS	OAX	OCEAN	ODE	OG	OGN
OKB	OM	OMG	OMNI	ONE	ONG
ONT	OOKI	OP	ORBS	ORCA	ORN
ORS	OSMO	OST	OXT	OXY	PASS
PAXG	PEOPLE	PERL	PERP	PHA	PHB
PHX	PIVX	PLA	PLANETS	PLU	PNG
PNK	PNT	POA	POE	POLIS	POLS
POLY	POLYX	POND	PORT	PORTO	POWR
PPT	PRISM	PRO	PROM	PROS	PSG
PTU	PUNDIX	PYR	QI	QKC	QLC
QNT	QRDO	QSP	QTF	QTUM	QUICK
RAD	RAI	RAMP	RARE	RARI	RAY
RBN	RBTC	RCN	RDN	REAL	REEF
REI	REN	RENBTC	REP	REQ	RGT
RIF	RLC	RLY	RNDR	RON	ROOK
ROSE	RRT	RSR	RUNE	RVN	SALT

Supported Cryptocurrency					
SAN	SAND	SANTOS	SC	SCRT	SFP
SGB	SHIB	SHPING	SKL	SKY	SLND
SLP	SLRS	SNGLS	SNM	SNT	SNX
SNY	SOL	SOS	SPARTA	SPELL	SPY
SRM	SSV	STARS	STEEM	STEP	STETH
STG	STMX	STORJ	STORM	STPT	STRAX
STSOL	STX	SUB	SUKU	SUN	SUPER
SUSD	SUSHI	SWRV	SXP	SYS	T
TBTC	TCT	TFUEL	THETA	TKO	TLM
TLOS	TNB	TNT	TOMO	TORN	TRAC
TRADE	TRB	TRIBE	TROY	TRU	TRX
TRYB	TSLA	TULIP	TUSD	TVK	TWT
UFT	UMA	UNFI	UNI	UOS	UPI
USDC	USDP	USDS	USDT	USTC	UTK
VAI	VEE	VELO	VEN	VET	VGX
VIA	VIB	VIBE	VIDT	VITE	VOXEL
VTHO	WABI	WAN	WAVES	WAXP	WBTC
WCFG	WILD	WIN	WING	WINGS	WNCG
WNDR	WNXM	WOO	WPR	WRX	WTC
XAUT	XCHF	XDC	XDG	XEC	XEM
XLM	XMR	XNO	XRA	XRD	XRP
XTZ	XVG	XVS	XYO	YFI	YFII
YGG	ZCN	ZEC	ZEN	ZIL	ZM
ZMT	ZRX				

Data Point Attributes

The Crypto Market Cap dataset provides **CoinGecko** objects, which have the following attributes:

Requesting Data

To add CoinGecko Crypto Market Cap data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
from Algorithm import *

class CoinGeckoMarketCapDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddCrypto("BTCUSD", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(CoinGecko, "BTC").Symbol
```

Accessing Data

To get the current Crypto Market Cap data, index the current **Slice** with the dataset **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} market cap::volume at {slice.Time}: {data_point.MarketCap}::
{data_point.Volume}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(CoinGecko).items():
        self.Log(f"{dataset_symbol} market cap::volume at {slice.Time}: {data_point.MarketCap}::
{data_point.Volume}")
```

Historical Data

To get historical CoinGecko Crypto Market Cap data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[CoinGecko](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Cryptos based on CoinGecko Crypto Market Cap data, call the **AddUniverse** method with the **CoinGeckoUniverse** class and a selection function. Note that the filtered output is a list of names of the

coins. If corresponding tradable crypto pairs are preferred, call **CreateSymbol(market, quoteCurrency)** method for each output item.

```
PY
def Initialize(self) -> None:
    self.AddUniverse(CoinGeckoUniverse, "CoinGeckoUniverse", Resolution.Daily, self.UniverseSelection)

def UniverseSelection(self, data: List[CoinGeckoUniverse]) -> List[Symbol]:
    for datum in data:
        self.Debug(f'{datum.Coin},{datum.MarketCap},{datum.Price}')

    # define our selection criteria
    selected = sorted(data, key=lambda x: x.MarketCap, reverse=True)[:3]

    # Use the CreateSymbol method to generate the Symbol object for
    # the desired market (Coinbase) and quote currency (e.g. USD)
    return [x.CreateSymbol(Market.GDAX, "USD") for x in selected]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove your subscription to CoinGecko Crypto Market Cap data, call the **RemoveSecurity** method.

```
PY
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The CoinGecko Crypto Market Cap dataset provide information on the size of the crypto coin and can be used to compare the size of one coin to another. Examples include the following strategies:

- Construct a major crypto index fund.
- Invest on the cryptos with the fastest growth in market size.
- Red flag stop when there might be a crypto bank run.

Datasets

CryptoSlam!

CryptoSlam! is an NFT industry data aggregator backed by Mark Cuban. Features project analytics, NFT values, rarity, scarcity, most popular collections, activity history & more.

NFT Sales

CryptoSlam!

NFT Sales

Introduction

The NFT Sales dataset by CryptoSlam! provides Non-Fungible Tokens (NFT) sales volume data in various blockchain marketplaces. This dataset covers 11 blockchains that have their own native Cryptocurrencies. The data starts in June 2017 and is delivered on a daily frequency. This dataset fetches the number of transactions, unique buyers, unique sellers, and the dollar volume of NFT transactions on all secondary marketplaces tracked by CryptoSlam, which includes owner-to-owner sales only (not initial sales from the product directly to the owners).

For more information about the NFT Sales dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CryptoSlam! is an NFT industry data aggregator backed by Mark Cuban. Features project analytics, NFT values, rarity, scarcity, most popular collections, activity history & more.

Getting Started

The following snippet demonstrates how to request data from the NFT Sales dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddCrypto("ETHUSD", Resolution.Daily, Market.Bitfinex).Symbol
self.dataset_symbol = self.AddData(CryptoSlamNFTSales, "ETH").Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	June 2017
Asset Coverage	11 blockchains
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Supported Blockchains and Symbols

The following table shows the blockchains tracked in the NFT Sales dataset:

Symbol	Blockchain Represented	Start Date (yyyy-mm-dd)
AVAX	Avalanche	2021-09-01
CRO	Cronos	2021-12-18
ETH	Ethereum	2017-06-23
FLOW	Flow	2020-07-28
FTM	Fantom	2021-09-15
MATIC	Polygon	2021-03-01
SOL	Solana	2021-08-05
THETA	Theta	2021-06-24
WAVE	Waves	2021-06-03
WAX	Wax	2020-03-16
XTZ	Tezos	2021-03-01

Data Point Attributes

The NFT Sales dataset provides **CryptoSlamNFTSales** objects, which have the following attributes:

Requesting Data

To add NFT Sales data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class CryptoSlamNFTSalesAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddCrypto("ETHUSD", Resolution.Daily, Market.Bitfinex).Symbol
        self.dataset_symbol = self.AddData(CryptoSlamNFTSales, "ETH").Symbol
```

Accessing Data

To get the current NFT Sales data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} unique buyers at {slice.Time}: {data_point.UniqueBuyers}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(CryptoSlamNFTSales).items():
        self.Log(f"{dataset_symbol} unique buyers at {slice.Time}: {data_point.UniqueBuyers}")
```

Historical Data

To get historical NFT Sales data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[CryptoSlamNFTSales](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

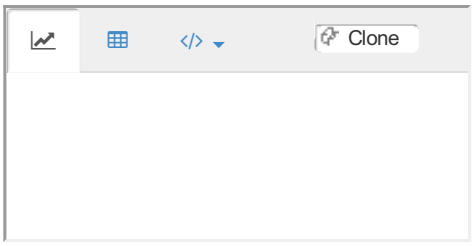
To remove a subscription, call the **RemoveSecurity** method.

```
PY
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The NFT Sales dataset enables you to incorporate NFT sales information into your strategies. Examples include the following strategies:

- Studying the correlation between the supply-demand trend of NFTs and the price changes of the underlying Cryptocurrencies.
- Measuring the activity/popularity blockchains to provide insight on the future price movements of the underlying Cryptocurrencies.


Clone

Datasets

Energy Information Administration

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation"s financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

US Energy Information Administration (EIA)

Energy Information Administration

US Energy Information Administration (EIA)

Introduction

The US Energy Information Administration (EIA) datasets by the Department of the Treasury tracks national and international oil production and consumption. The data covers 190 datasets, starts in January 1991, and is delivered on a daily frequency. This dataset is created by QuantConnect processing and caching the EIA archives.

For more information about the US Energy Information Administration (EIA) dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation's financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

Getting Started

The following snippet demonstrates how to request data from the EIA dataset:

```py
from QuantConnect.DataSource import *

self.dataset_symbol = self.AddData(USEnergy,
USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1991
Asset Coverage	190 Datasets
Data Density	Sparse
Resolution	Daily
Timezone	New York

Supported Datasets

The following table shows the accessor code you need to add each EIA dataset to your algorithm:

Symbol	Accessor Code	Description
UnitedStates		
PET.WGFRPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderAdjustedNetProductionOfFinishedMotorGasoline`	U.S. Refiner and Blender Adjusted Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGFSTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfFinishedMotorGasoline`	U.S. Ending Stocks of Finished Motor Gasoline in Thousand Barrels (Mbbl)
PET.WGFUPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfFinishedMotorGasoline`	U.S. Product Supplied of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WCSSTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfCrudeOilInSpr`	U.S. Ending Stocks of Crude Oil in SPR in Thousand Barrels (Mbbl)
PET.WDGRPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfDistillateFuelOilGreaterThan500PpmSulfur`	U.S. Refiner and Blender Net Production of Distillate Fuel Oil Greater than 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WDGSTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOilGreaterThan500PpmSulfur`	U.S. Ending Stocks of Distillate Fuel Oil, Greater Than 500 ppm Sulfur in Thousand Barrels (Mbbl)
PET.WDIEXUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyExportsOfTotalDistillate`	U.S. Exports of Total Distillate in Thousand Barrels per Day (Mbbl/d)
PET.WDIIMUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOil`	U.S. Imports of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WDIRPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfDistillateFuelOil`	U.S. Refiner and Blender Net Production of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.WKJSTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfKeroseneTypeJetFuel`	U.S. Ending Stocks of Kerosene-Type Jet Fuel in Thousand Barrels (Mbbl)
PET.WKJUPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfKeroseneTypeJetFuel`	U.S. Product Supplied of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WGTIMUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyImportsOfTotalGasoline`	U.S. Imports of Total Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGTSTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfTotalGasoline`	U.S. Ending Stocks of Total Gasoline in Thousand Barrels (Mbbl)
PET.WGIRIUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyGrossInputsIntoRefineries`	U.S. Gross Inputs into Refineries in Thousand Barrels per Day (Mbbl/d)
PET.WGRIMUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedMotorGasoline`	U.S. Imports of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGRRPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfReformulatedMotorGasoline`	U.S. Refiner and Blender Net Production of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGRSTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedMotorGasoline`	U.S. Ending Stocks of Reformulated Motor Gasoline in Thousand Barrels (Mbbl)
PET.WDISTUS1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOil`	U.S. Ending Stocks of Distillate Fuel Oil in Thousand Barrels (Mbbl)
PET.WDIUPUS2.W	`USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfDistillateFuelOil`	U.S. Product Supplied of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.WKMRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfMilitaryKeroseneTypeJetFuel	U.S. Refiner and Blender Net Production of Military Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WOCLEUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyOperableCrudeOilDistillationCapacity	U. S. Operable Crude Oil Distillation Capacity in Thousand Barrels per Calendar Day (Mbbl/d)
PET.WPLSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyPropyleneNonfuelUseStocksAtBulkTerminals	U.S. Propylene Nonfuel Use Stocks at Bulk Terminals in Thousand Barrels (Mbbl)
PET.WPRSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfPropaneAndPropylene	U.S. Ending Stocks of Propane and Propylene in Thousand Barrels (Mbbl)
PET.WPULEUS3.W	USEnergy.Petroleum.UnitedStates.WeeklyPercentUtilizationOfRefineryOperableCapacity	U.S. Percent Utilization of Refinery Operable Capacity in Percent (%)
PET.WREEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfResidualFuelOil	U.S. Exports of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WREIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfResidualFuelOil	U.S. Imports of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WKCRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfCommercialKeroseneTypeJetFuel	U.S. Refiner and Blender Net Production of Commercial Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WKJEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfKeroseneTypeJetFuel	U.S. Exports of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.WKJIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfKeroseneTypeJetFuel	U.S. Imports of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WKJRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfKeroseneTypeJetFuel	U.S. Refiner and Blender Net Production of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WCESTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksExcludingSprOfCrudeOil	U.S. Ending Stocks excluding SPR of Crude Oil in Thousand Barrels (Mbbl)
PET.WCREXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfCrudeOil	U.S. Exports of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRFPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyFieldProductionOfCrudeOil	U.S. Field Production of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfCrudeOil	U.S. Imports of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRNTUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfCrudeOil	U.S. Net Imports of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRRIUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetInputOfCrudeOil	U.S. Refiner Net Input of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WRERPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfResidualFuelOil	U.S. Refiner and Blender Net Production of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WRESTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfResidualFuelOil	U.S. Ending Stocks of Residual Fuel Oil in Thousand Barrels (Mbbl)
PET.WREUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfResidualFuelOil	U.S. Product Supplied of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.WRPEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfTotalPetroleumProducts	U.S. Exports of Total Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WRPIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfTotalPetroleumProducts	U.S. Imports of Total Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WRPNTUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts	U.S. Net Imports of Total Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WRPUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfPetroleumProducts	U.S. Product Supplied of Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTESTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksExcludingSprOfCrudeOilAndPetroleumProducts	U.S. Ending Stocks excluding SPR of Crude Oil and Petroleum Products in Thousand Barrels (Mbbl)
PET.WTTEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfCrudeOilAndPetroleumProducts	U.S. Exports of Crude Oil and Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTTIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfCrudeOilAndPetroleumProducts	U.S. Imports of Crude Oil and Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTTNTUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfCrudeOilAndPetroleumProducts	U.S. Net Imports of Crude Oil and Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTTSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfCrudeOilAndPetroleumProducts	U.S. Ending Stocks of Crude Oil and Petroleum Products in Thousand Barrels (Mbbl)
PET.WUOSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfUnfinishedOils	U.S. Ending Stocks of Unfinished Oils in Thousand Barrels (Mbbl)

Symbol	Accessor Code	Description
PET.WG6TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfOtherFinishedConventionalMotorGasoline	U.S. Refiner and Blender Net Production of Other Finished Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WD0TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfDistillateFuelOil0To15PpmSulfur	U.S. Refiner and Blender Net Production of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WD1ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Ending Stocks of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels (Mbbl)
PET.WD1TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductionOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Production of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WG1ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Ending Stocks of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels (Mbbl)
PET.WCRSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfCrudeOil	U.S. Ending Stocks of Crude Oil in Thousand Barrels (Mbbl)
PET.WCSIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyCrudeOilImportsBySpr	U.S. Crude Oil Imports by SPR in Thousand Barrels per Day (Mbbl/d)
PET.WBCIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfGasolineBlendingComponents	U.S. Imports of Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WBCSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfGasolineBlendingComponents	U.S. Ending Stocks of Gasoline Blending Components in Thousand Barrels (Mbbl)

Symbol	Accessor Code	Description
PET.WCEIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyCommercialCrudeOilImportsExcludingSpr	U.S. Commercial Crude Oil Imports Excluding SPR in Thousand Barrels per Day (Mbbl/d)
PET.WPRTP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerBlenderAndGasPlantNetProductionOfPropaneAndPropylene	U.S. Refiner, Blender, and Gas Plant Net Production of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.WG1TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedReformulatedMotorGasolineWithEthanol	U.S. Refiner and Blender Net Production of Finished Reformulated Motor Gasoline with Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.WG3ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedMotorGasolineNonOxygentated	U.S. Ending Stocks of Reformulated Motor Gasoline, Non-Oxygentated in Thousand Barrels (Mbbl)
PET.WG4ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasoline	U.S. Ending Stocks of Conventional Motor Gasoline in Thousand Barrels (Mbbl)
PET.WG4TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfConventionalMotorGasoline	U.S. Refiner and Blender Net Production of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WG5ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasolineWithFuelEthanol	U.S. Ending Stocks of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels (Mbbl)
PET.WG5TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedConventionalMotorGasolineWithEthanol	U.S. Refiner and Blender Net Production of Finished Conventional Motor Gasoline with Ethanol in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.WG6ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfOtherConventionalMotorGasoline	U.S. Ending Stocks of Other Conventional Motor Gasoline in Thousand Barrels (Mbbl)
PET.WO6RI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfConventionalCbobGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Conventional CBOB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO6ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalCbobGasolineBlendingComponents	U.S. Ending Stocks of Conventional CBOB Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO7RI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfConventionalGtabGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Conventional GTAB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO7ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalGtabGasolineBlendingComponents	U.S. Ending Stocks of Conventional GTAB Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO9RI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfConventionalOtherGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Conventional Other Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO9ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalOtherGasolineBlendingComponents	U.S. Ending Stocks of Conventional Other Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.W_EPD2F_PWR_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2HeatingOilWholesaleResalePrice	U.S. No. 2 Heating Oil Wholesale/Resale Price in Dollars per Gallon ($/gal)

Symbol	Accessor Code	Description
PET.W_EPC0_SKA_NUS_MBBL.W	`USEnergy.Petroleum.UnitedStates.WeeklyCrudeOilStocksInTransitOnShipsFromAlaska`	U.S. Crude Oil Stocks in Transit (on Ships) from Alaska in Thousand Barrels (Mbbl)
PET.W_EPC0_VSD_NUS_DAYS.W	`USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfCrudeOilExcludingSpr`	U.S. Days of Supply of Crude Oil excluding SPR in Number of Days (Days)
PET.W_EPD0_VSD_NUS_DAYS.W	`USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfTotalDistillate`	U.S. Days of Supply of Total Distillate in Number of Days (Days)
PET.W_EPD2F_PRS_NUS_DPG.W	`USEnergy.Petroleum.UnitedStates.WeeklyWeeklyNo2HeatingOilResidentialPrice`	U.S. Weekly No. 2 Heating Oil Residential Price in Dollars per Gallon ($/gal)
PET.WPRUP_NUS_2.W	`USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfPropaneAndPropylene`	U.S. Product Supplied of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.WWOUP_NUS_2.W	`USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfOtherOils`	U.S. Product Supplied of Other Oils in Thousand Barrels per Day (Mbbl/d)
PET.WBCRI_NUS_2.W	`USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfGasolineBlendingComponents`	U.S. Refiner and Blender Net Input of Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WD0ST_NUS_1.W	`USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOil0To15PpmSulfur`	U.S. Ending Stocks of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels (Mbbl)
PET.W_EPJK_VSD_NUS_DAYS.W	`USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfKeroseneTypeJetFuel`	U.S. Days of Supply of Kerosene-Type Jet Fuel in Number of Days (Days)
PET.W_EPM0_VSD_NUS_DAYS.W	`USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfTotalGasoline`	U.S. Days of Supply of Total Gasoline in Number of Days (Days)

Symbol	Accessor Code	Description
PET.W_EPPA_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfAsphaltAndRoadOil	U.S. Ending Stocks of Asphalt and Road Oil in Thousand Barrels (Mbbl)
PET.W_EPPK_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfKerosene	U.S. Ending Stocks of Kerosene in Thousand Barrels (Mbbl)
PET.W_EPDM10_VUA_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklySupplyAdjustmentOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Supply Adjustment of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WG5IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalMotorGasolineWithFuelEthanol	U.S. Imports of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.WG6IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfOtherConventionalMotorGasoline	U.S. Imports of Other Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WD0IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOil0To15PpmSulfur	U.S. Imports of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WD1IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Imports of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WD2IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOilGreaterThan500To2000PpmSulfur	U.S. Imports of Distillate Fuel Oil, Greater than 500 to 2000 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WPRIM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfPropaneAndPropylene	U.S. Imports of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.WO7IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalGtabGasolineBlendingComponents	U.S. Imports of Conventional GTAB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WD3IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOilGreaterThan2000PpmSulfur	U.S. Imports of Distillate Fuel Oil, Greater than 2000 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WG1IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Imports of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels per Day (Mbbl/d)
PET.WG4IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalMotorGasoline	U.S. Imports of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WO9IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalOtherGasolineBlendingComponents	U.S. Imports of Conventional Other Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO6IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalCbobGasolineBlendingComponents	U.S. Imports of Conventional CBOB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPK_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfKerosene	U.S. Blender Net Production of Kerosene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPK_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfKerosene	U.S. Refiner Net Production of Kerosene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPO6_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfOtherOilsExcludingFuelEthanol	U.S. Ending Stocks of Other Oils (Excluding Fuel Ethanol) in Thousand Barrels (Mbbl)
PET.W_EPPR_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfResidualFuelOil	U.S. Refiner Net Production of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPM0R_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfReformulatedMotorGasoline	U.S. Blender Net Production of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0R_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfReformulatedMotorGasoline	U.S. Refiner Net Production of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOOXE_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfFuelEthanol	U.S. Ending Stocks of Fuel Ethanol in Thousand Barrels (Mbbl)
PET.W_EPD0_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOil	U.S. Blender Net Production of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.W_EPD0_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOil	U.S. Refiner Net Production of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.W_EPJK_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfKeroseneTypeJetFuel	U.S. Blender Net Production of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.W_EPJK_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfKeroseneTypeJetFuel	U.S. Refiner Net Production of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.W_EPLLPA_PRS_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPropaneResidentialPrice	U.S. Propane Residential Price in Dollars per Gallon ($/gal)
PET.W_EPLLPA_PWR_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPropaneWholesaleResalePrice	U.S. Propane Wholesale/Resale Price in Dollars per Gallon ($/gal)
PET.W_EPOBGRR_YIR_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfMotorGasolineBlendingComponentsRbob	U.S. Refiner and Blender Net Input of Motor Gasoline Blending Components, RBOB in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPL0XP_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfNgplsLrgsExcludingPropanePropylene	U.S. Ending Stocks of NGPLs/LRGs (Excluding Propane/Propylene) in Thousand Barrels (Mbbl)
PET.W_EPLLPZ_VSD_NUS_DAYS.W	USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfPropanePropylene	U.S. Days of Supply of Propane/Propylene in Number of Days (Days)
PET.W_EPM0C_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfConventionalMotorGasoline	U.S. Blender Net Production of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0C_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfConventionalMotorGasoline	U.S. Refiner Net Production of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_VUA_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklySupplyAdjustmentOfFinishedMotorGasoline	U.S. Supply Adjustment of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfFinishedMotorGasoline	U.S. Blender Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_YPR_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedMotorGasoline	U.S. Refiner and Blender Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfFinishedMotorGasoline	U.S. Refiner Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPD00H_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOilGreaterThan500PpmSulfur	U.S. Blender Net Production of Distillate Fuel Oil, Greater Than 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPD00H_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOilGreaterThan500PpmSulfur	U.S. Refiner Net Production of Distillate Fuel Oil, Greater Than 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDM10_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Blender Net Production of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDM10_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Refiner Net Production of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDXL0_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOil0To15PpmSulfur	U.S. Blender Net Production of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDXL0_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOil0To15PpmSulfur	U.S. Refiner Net Production of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CA_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfConventionalMotorGasolineWithFuelEthanol	U.S. Blender Net Production of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CA_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfConventionalMotorGasolineWithFuelEthanol	U.S. Refiner Net Production of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CO_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfOtherConventionalMotorGasoline	U.S. Blender Net Production of Other Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPM0CO_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfOtherConventionalMotorGasoline	U.S. Refiner Net Production of Other Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RA_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Blender Net Production of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RA_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Refiner Net Production of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOOXE_YOP_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyOxygenatePlantProductionOfFuelEthanol	U.S. Oxygenate Plant Production of Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfMotorGasolineFinishedConventionalEd55AndLower	U.S. Blender Net Production of Motor Gasoline, Finished, Conventional, Ed55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_YPT_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedConventionalMotorGasolineEd55AndLower	U.S. Refiner and Blender Net Production of Finished Conventional Motor Gasoline, Ed 55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfMotorGasolineFinishedConventionalEd55AndLower	U.S. Refiner Net Production of Motor Gasoline, Finished, Conventional, Ed55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_EEX_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfFinishedMotorGasoline	U.S. Exports of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPM0F_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfFinishedMotorGasoline	U.S. Imports of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfOtherReformulatedMotorGasoline	U.S. Blender Net Production of Other Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_YPT_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfOtherFinishedReformulatedMotorGasoline	U.S. Refiner and Blender Net Production of Other Finished Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfOtherReformulatedMotorGasoline	U.S. Refiner Net Production of Other Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOBGRR_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfMotorGasolineBlendingComponentsRbob	U.S. Ending Stocks of Motor Gasoline Blending Components, RBOB in Thousand Barrels (Mbbl)
PET.W_EPOOXE_YIR_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfFuelEthanol	U.S. Refiner and Blender Net Input of Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfMotorGasolineFinishedConventionalGreaterThanEd55	U.S. Imports of Motor Gasoline, Finished, Conventional, Greater than Ed55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfMotorGasolineFinishedConventionalEd55AndLower	U.S. Imports of Motor Gasoline, Finished, Conventional, Ed55 and Lower in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPC0_IMU_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyCrudeOilImportsForSprByOthers	U.S. Crude Oil Imports for SPR by Others in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasolineGreaterThanEd55	U.S. Ending Stocks of Conventional Motor Gasoline, Greater than Ed55 in Thousand Barrels (Mbbl)
PET.W_EPOOXE_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfFuelEthanol	U.S. Imports of Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPL0XP_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfLiquefiedPetroleumGassesLessPropanePropylene	U.S. Imports of Liquefied Petroleum Gasses Less Propane/Propylene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPLLPZ_EEX_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfPropaneAndPropylene	U.S. Exports of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfOtherReformulatedMotorGasoline	U.S. Imports of Other Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfMotorGasolineFinishedConventionalGreaterThanEd55	U.S. Blender Net Production of Motor Gasoline, Finished, Conventional, Greater Than Ed55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_YPT_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedConventionalMotorGasolineGreaterThanEd55	U.S. Refiner and Blender Net Production of Finished Conventional Motor Gasoline, Greater than Ed 55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfFinishedConventionalMotorGasolineGreaterThanEd55	U.S. Refiner Net Production of Finished Conventional Motor Gasoline, Greater than Ed 55 in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
PET.W_EPM0CAL55_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasolineEd55AndLower	U.S. Ending Stocks of Conventional Motor Gasoline, Ed55 and Lower in Thousand Barrels (Mbbl)
PET.W_EPPK_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfKerosene	U.S. Imports of Kerosene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPO4_EEX_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfOtherOils	U.S. Exports of Other Oils in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPO6_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfOtherOilsExcludingFuelEthanol	U.S. Imports of Other Oils (Excluding Fuel Ethanol) in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOBGRR_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsFromAllCountriesOfMotorGasolineBlendingComponentsRbob	U.S. Imports from All Countries of Motor Gasoline Blending Components, RBOB in Thousand Barrels per Day (Mbbl/d)
PET.EMM_EPMR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyRegularAllFormulationsRetailGasolinePrices	U.S. Regular All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMM_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyMidgradeAllFormulationsRetailGasolinePrices	U.S. Midgrade All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMP_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPremiumAllFormulationsRetailGasolinePrices	U.S. Premium All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPM0_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyAllGradesAllFormulationsRetailGasolinePrices	U.S. All Grades All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPM0R_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyAllGradesReformulatedRetailGasolinePrices	U.S. All Grades Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)

Symbol	Accessor Code	Description
PET.EMM_EPMMR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyMidgradeReformulatedRetailGasolinePrices	U.S. Midgrade Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMPR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPremiumReformulatedRetailGasolinePrices	U.S. Premium Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMRU_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyRegularConventionalRetailGasolinePrices	U.S. Regular Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMRR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyRegularReformulatedRetailGasolinePrices	U.S. Regular Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMD_EPD2D_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2DieselRetailPrices	U.S. No 2 Diesel Retail Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMPU_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPremiumConventionalRetailGasolinePrices	U.S. Premium Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMMU_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyMidgradeConventionalRetailGasolinePrices	U.S. Midgrade Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPM0U_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyAllGradesConventionalRetailGasolinePrices	U.S. All Grades Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMD_EPD2DXL0_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2DieselUltraLowSulfur015PpmRetailPrices	U.S. No 2 Diesel Ultra Low Sulfur (0-15 ppm) Retail Prices in Dollars per Gallon ($/gal)
PET.W_EPC0_SAX_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksExcludingSprAndIncludingLeaseStockOfCrudeOil	U.S. Ending Stocks excluding SPR and including Lease Stock of Crude Oil in Thousand Barrels (Mbbl)

Symbol	Accessor Code	Description
PET.EMD_EPD2DM10_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2DieselLowSulfur15500PpmRetailPrices	U.S. No 2 Diesel Low Sulfur (15-500 ppm) Retail Prices in Dollars per Gallon ($/gal)
PET.WO3IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedRbobWithAlcoholGasolineBlendingComponents	U.S. Imports of Reformulated RBOB with Alcohol Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO4IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedRbobWithEtherGasolineBlendingComponents	U.S. Imports of Reformulated RBOB with Ether Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO2ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedGtabGasolineBlendingComponents	U.S. Ending Stocks of Reformulated GTAB Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO3ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedRbobWithAlcoholGasolineBlendingComponents	U.S. Ending Stocks of Reformulated RBOB with Alcohol Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO4ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedRbobWithEtherGasolineBlendingComponents	U.S. Ending Stocks of Reformulated RBOB with Ether Gasoline Blending Components in Thousand Barrels (Mbbl)
EquatorialGuinea		
PET.W_EPC0_IM0_NUS-NEK_MBBLD.W	USEnergy.Petroleum.EquatorialGuinea.WeeklyImportsFromEquatorialGuineaOfCrudeOil	U.S. Imports from Equatorial Guinea of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Iraq		
PET.W_EPC0_IM0_NUS-NIZ_MBBLD.W	USEnergy.Petroleum.Iraq.WeeklyImportsFromIraqOfCrudeOil	U.S. Imports from Iraq of Crude Oil in Thousand Barrels per Day (Mbbl/d)

Symbol	Accessor Code	Description
Kuwait		
PET.W_EPC0_IM0_NUS-NKU_MBBLD.W	`USEnergy.Petroleum.Kuwait.WeeklyImportsFromKuwaitOfCrudeOil`	U.S. Imports from Kuwait of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Mexico		
PET.W_EPC0_IM0_NUS-NMX_MBBLD.W	`USEnergy.Petroleum.Mexico.WeeklyImportsFromMexicoOfCrudeOil`	U.S. Imports from Mexico of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Nigeria		
PET.W_EPC0_IM0_NUS-NNI_MBBLD.W	`USEnergy.Petroleum.Nigeria.WeeklyImportsFromNigeriaOfCrudeOil`	U.S. Imports from Nigeria of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Norway		
PET.W_EPC0_IM0_NUS-NNO_MBBLD.W	`USEnergy.Petroleum.Norway.WeeklyImportsFromNorwayOfCrudeOil`	U.S. Imports from Norway of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Russia		
PET.W_EPC0_IM0_NUS-NRS_MBBLD.W	`USEnergy.Petroleum.Russia.WeeklyImportsFromRussiaOfCrudeOil`	U.S. Imports from Russia of Crude Oil in Thousand Barrels per Day (Mbbl/d)
SaudiArabia		
PET.W_EPC0_IM0_NUS-NSA_MBBLD.W	`USEnergy.Petroleum.SaudiArabia.WeeklyImportsFromSaudiArabiaOfCrudeOil`	U.S. Imports from Saudi Arabia of Crude Oil in Thousand Barrels per Day (Mbbl/d)
UnitedKingdom		
PET.W_EPC0_IM0_NUS-NUK_MBBLD.W	`USEnergy.Petroleum.UnitedKingdom.WeeklyImportsFromUnitedKingdomOfCrudeOil`	U.S. Imports from United Kingdom of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Venezuela		

Symbol	Accessor Code	Description
PET.W_EPC0_IM0_NUS-NVE_MBBLD.W	USEnergy.Petroleum.Venezuela.WeeklyImportsFromVenezuelaOfCrudeOil	U.S. Imports from Venezuela of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Algeria		
PET.W_EPC0_IM0_NUS-NAG_MBBLD.W	USEnergy.Petroleum.Algeria.WeeklyImportsFromAlgeriaOfCrudeOil	U.S. Imports from Algeria of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Angola		
PET.W_EPC0_IM0_NUS-NAO_MBBLD.W	USEnergy.Petroleum.Angola.WeeklyImportsFromAngolaOfCrudeOil	U.S. Imports from Angola of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Brazil		
PET.W_EPC0_IM0_NUS-NBR_MBBLD.W	USEnergy.Petroleum.Brazil.WeeklyImportsFromBrazilOfCrudeOil	U.S. Imports from Brazil of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Canada		
PET.W_EPC0_IM0_NUS-NCA_MBBLD.W	USEnergy.Petroleum.Canada.WeeklyImportsFromCanadaOfCrudeOil	U.S. Imports from Canada of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Congo		
PET.W_EPC0_IM0_NUS-NCF_MBBLD.W	USEnergy.Petroleum.Congo.WeeklyImportsFromCongoBrazzavilleOfCrudeOil	U.S. Imports from Congo (Brazzaville) of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Colombia		
PET.W_EPC0_IM0_NUS-NCO_MBBLD.W	USEnergy.Petroleum.Colombia.WeeklyImportsFromColombiaOfCrudeOil	U.S. Imports from Colombia of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Ecuador		
PET.W_EPC0_IM0_NUS-NEC_MBBLD.W	USEnergy.Petroleum.Ecuador.WeeklyImportsFromEcuadorOfCrudeOil	U.S. Imports from Ecuador of Crude Oil in Thousand Barrels per Day (Mbbl/d)

Data Point Attributes

The EIA dataset provides **USEnergy** objects, which have the following attributes:

Requesting Data

To add EIA data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class USEnergyDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.dataset_symbol = self.AddData(USEnergy,
USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts).Symbol
```

Accessing Data

To get the current EIA data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} value at {slice.Time}: {data_point.Value}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(USEnergy).items():
        self.Log(f"{dataset_symbol} value at {slice.Time}: {data_point.Value}")
```

Historical Data

To get historical EIA data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[USEnergy](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

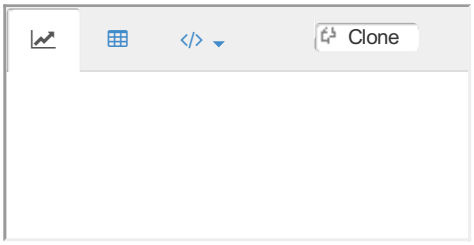
To remove your subscription to EIA data, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The EIA dataset enables you to monitor national and international oil production and consumption in you trading strategies. Examples include the following strategies:

- Trading petroleum companies when there is a change in net imports of petroleum products
- Trading country ETFs when there is a change in the country's net import of resources
- Adjusting exposure to vehicle manufacturer stocks when the supply of gasoline is higher/lower than historical levels



Datasets

ExtractAlpha

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha"s rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Cross Asset Model

Estimize

Tactical

True Beats

ExtractAlpha

Cross Asset Model

Introduction

The Cross Asset Model by ExtractAlpha provides stock scoring values based on the trading activity in the Options market. Since the Options market has a higher proportion of institutional traders than the Equities market, the Options market is composed of investors who are more informed and information-driven on average. The data covers a dynamic universe of over 3,000 US Equities, starts in July 2005, and is delivered on a daily frequency. This dataset is created by feature engineering on the Options market put-call spread, volatility skewness, and volume.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Cross Asset Model dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the Cross Asset Model dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(ExtractAlphaCrossAssetModel, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 2005
Asset Coverage	Over 3,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Cross Asset Model dataset provides **ExtractAlphaCrossAssetModel** objects, which have the following attributes:

Requesting Data

To add Cross Asset Model data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class ExtractAlphaCrossAssetModelDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(ExtractAlphaCrossAssetModel, self.symbol).Symbol
```

Accessing Data

To get the current Cross Asset Model data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} score at {slice.Time}: {data_point.Score}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(ExtractAlphaCrossAssetModel).items():
        self.Log(f"{dataset_symbol} score at {slice.Time}: {data_point.Score}")
```

Historical Data

To get historical Cross Asset Model data, call the **History** method with the dataset **Symbol** . If there is no data in the

period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[ExtractAlphaCrossAssetModel](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

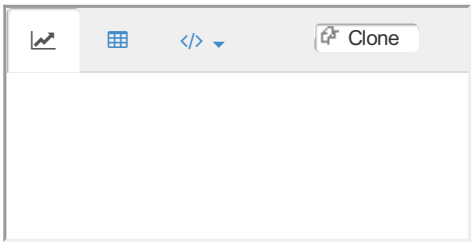
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Cross Asset Model data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Cross Asset Model dataset by ExtractAlpha enables you to utilize Options market information to extract alpha. Examples include the following strategies:

- Predicting price and volatility changes in Equities.
- Signaling arbitrage opportunities between Options and underlying assets.
- Using it as a stock selection indicator.



ExtractAlpha

Estimize

Introduction

The Estimize dataset by ExtractAlpha estimates the financials of companies, including EPS, revenues, industry-specific KPIs, macroeconomic indicators, and more. The data covers over 2,800 US-listed Equities' EPS/Revenue, over 200 company KPIs, 27 US and 55 international macroeconomic indicator datasets, and more. The data starts in January 2011 and is delivered on a daily frequency. This dataset is crowdsourced from a community of 100,000+ contributors via the data provider's web platform.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Estimize dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the Estimize dataset:

```
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.estimize_consensus_symbol = self.AddData(EstimizeConsensus, self.symbol).Symbol
self.estimize_estimate_symbol = self.AddData(EstimizeEstimate, self.symbol).Symbol
self.estimize_release_symbol = self.AddData(EstimizeRelease, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2011
Asset Coverage	2,800 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Estimize dataset provides **EstimizeConsensus** , **EstimizeEstimate** , and **EstimizeRelease** objects.

EstimizeConsensus Attributes

EstimizeConsensus objects have the following attributes:

EstimizeEstimate Attributes

EstimizeEstimate objects have the following attributes:

EstimizeRelease Attributes

EstimizeRelease objects have the following attributes:

Requesting Data

To add Estimize data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class ExtractAlphaEstimizeDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.estimize_consensus_symbol = self.AddData(EstimizeConsensus, self.symbol).Symbol
        self.estimize_estimate_symbol = self.AddData(EstimizeEstimate, self.symbol).Symbol
        self.estimize_release_symbol = self.AddData(EstimizeRelease, self.symbol).Symbol
```

Accessing Data

To get the current Estimize data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.estimize_consensus_symbol):
        data_point = slice[self.estimize_consensus_symbol]
        self.Log(f"{self.estimize_consensus_symbol} mean at {slice.Time}: {data_point.Mean}")

    if slice.ContainsKey(self.estimize_estimate_symbol):
        data_point = slice[self.estimize_estimate_symbol]
        self.Log(f"{self.estimize_estimate_symbol} EPS at {slice.Time}: {data_point.Eps}")

    if slice.ContainsKey(self.estimize_release_symbol):
        data_point = slice[self.estimize_release_symbol]
        self.Log(f"{self.estimize_release_symbol} EPS at {slice.Time}: {data_point.Eps}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(EstimizeConsensus).items():
        self.Log(f"{dataset_symbol} mean at {slice.Time}: {data_point.Mentions}")

    for dataset_symbol, data_point in slice.Get(EstimizeEstimate).items():
        self.Log(f"{dataset_symbol} EPS at {slice.Time}: {data_point.Eps}")

    for dataset_symbol, data_point in slice.Get(EstimizeRelease).items():
        self.Log(f"{dataset_symbol} EPS at {slice.Time}: {data_point.Eps}")
```

Historical Data

To get historical Estimize data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrames
consensus_history_df = self.History(self.estimize_consensus_symbol, 100, Resolution.Daily)
estimate_history_df = self.History(self.estimize_estimate_symbol, 100, Resolution.Daily)
release_history_df = self.History(self.estimize_release_symbol, 100, Resolution.Daily)
history_df = self.History([
    self.estimize_consensus_symbol,
    self.estimize_estimate_symbol,
    self.estimize_release_symbol], 100, Resolution.Daily)

# Dataset objects
consensus_history_bars = self.History[EstimizeConsensus](self.estimize_consensus_symbol, 100,
Resolution.Daily)
estimate_history_bars = self.History[EstimizeEstimate](self.estimize_estimate_symbol, 100,
Resolution.Daily)
release_history_bars = self.History[EstimizeRelease](self.estimize_release_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

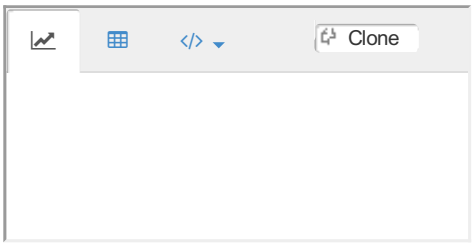
```
self.RemoveSecurity(self.estimize_consensus_symbol)
self.RemoveSecurity(self.estimize_estimate_symbol)
self.RemoveSecurity(self.estimize_release_symbol)
```

If you subscribe to Estimize data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Estimize dataset enables you to estimate the financial data of a company more accurately for alpha. Examples include the following use cases:

- Fundamental estimates for ML regression/classification models
- Arbitrage/Sentiment trading on market “surprise” from ordinary expectations based on the better expectation by the dataset
- Using industry-specific KPIs to predict the returns of individual sectors



ExtractAlpha

Tactical

Introduction

The Tactical dataset by ExtractAlpha is a stock scoring algorithm that captures the technical dynamics of individual US Equities over one to ten trading day horizons. It can assist a longer-horizon investor in timing their entry or exit points or be used in combination with existing systematic or qualitative strategies with similar holding periods.

The data covers a dynamic universe of around 4,700 US Equities per day on average, starts in January 2000, and is delivered on a daily frequency. The Tactical dataset expands upon simple reversal, liquidity, and seasonality factors to identify stocks that are likely to trend or reverse.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Tactical dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the Tactical dataset:

```py
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(ExtractAlphaTacticalModel, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2000
Asset Coverage	5,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Tactical dataset provides **ExtractAlphaTacticalModel** objects, which have the following attributes:

Requesting Data

To add Tactical data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class ExtractAlphaTacticalModelDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(ExtractAlphaTacticalModel, self.symbol).Symbol
```

Accessing Data

To get the current Tactical data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} score at {slice.Time}: {data_point.Score}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(ExtractAlphaTacticalModel).items():
        self.Log(f"{dataset_symbol} score at {slice.Time}: {data_point.Score}")
```

Historical Data

To get historical Tactical data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_df = self.History[ExtractAlphaTacticalModel](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

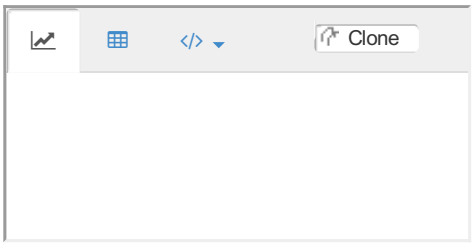
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Tactical data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Tactical dataset enables you to gain insight into short-term stock dynamics for trading. Examples include the following strategies:

- Optimizing entry and exit times in a portfolio construction model.
- Using the raw factor values as technical indicators.
- Inputting the data into machine learning classifier models as trend/reversal labels.



ExtractAlpha

True Beats

Introduction

The True Beats dataset by ExtractAlpha provides quantitative predictions of EPS and Revenues for US Equities. The data covers a dynamic universe of around 4,000-5,000 US-listed Equities on a daily average. The data starts in January 2000 and is delivered on a daily frequency. This dataset is created by incorporating the opinions of expert analysts, historical earnings, revenue trends for the company and its peers, and metrics on company earnings management.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the True Beats dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the True Beats dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(ExtractAlphaTrueBeats, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2000
Asset Coverage	Over 5,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The True Beats dataset provides **ExtractAlphaTrueBeat** objects, which have the following attributes:

The True Beats dataset provides **ExtractAlphaTrueBeats** and **ExtractAlphaTrueBeat** objects.

ExtractAlphaTrueBeats

ExtractAlphaTrueBeats objects have the following attributes:

ExtractAlphaTrueBeat

ExtractAlphaTrueBeat objects have the following attributes:

Requesting Data

To add True Beats data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY

class ExtractAlphaTrueBeatsDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(ExtractAlphaTrueBeats, self.symbol).Symbol
```

Accessing Data

To get the current True Beats data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} True beat at {slice.Time}: {data_point.TrueBeat}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY

def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(ExtractAlphaTrueBeats).items():
        self.Log(f"{dataset_symbol} True beat at {slice.Time}: {data_point.TrueBeat}")
```

Historical Data

To get historical True Beats data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[ExtractAlphaTrueBeats](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

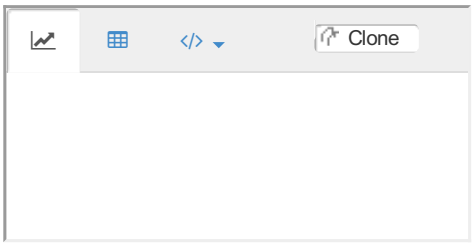
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to True Beats data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The True Beats dataset enables you to predict EPS and revenue of US-listed Equities for trading. Examples include the following strategies:

- Finding surprise in EPS or revenue for sentiment/arbitrage trading
- Stock or sector selections based on EPS or revenue predictions
- Calculate expected return by valuation models based on EPS or revenue predictions (e.g. Black-Litterman)



Datasets

FRED

The Research Division of the Federal Reserve bank of St. Louis, MO expands the frontier of economic knowledge by producing high-quality original research in the areas of macroeconomics, money and banking, and applied microeconomics. They contribute to monetary policy discussions by advising the Bank president on a range of topics, especially in preparation for Federal Open Market Committee (FOMC) meetings. The Research Division is in the top 1% of all economics research departments worldwide.

US Federal Reserve (FRED)

FRED

US Federal Reserve (FRED)

Introduction

The Federal Reserve Economic Data (FRED) by the Research Division of the Federal Reserve bank of St. Louis, MO provides various time series relating to macro-economic data. The data covers 560 datasets, starts in January 1999, and is delivered on a daily frequency. The data is created by aggregating daily updates from more than 85 public and proprietary sources.

For more information about the US Federal Reserve (FRED) dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Research Division of the Federal Reserve bank of St. Louis, MO expands the frontier of economic knowledge by producing high-quality original research in the areas of macroeconomics, money and banking, and applied microeconomics. They contribute to monetary policy discussions by advising the Bank president on a range of topics, especially in preparation for Federal Open Market Committee (FOMC) meetings. The Research Division is in the top 1% of all economics research departments worldwide.

Getting Started

The following snippet demonstrates how to request data from the FRED dataset:

```py
from QuantConnect.DataSource import *

self.dataset_symbol = self.AddData(Fred,
Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough, Resolution.Daily).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1990
Asset Coverage	560 Datasets
Data Density	Sparse
Resolution	Daily
Timezone	New York

Supported Datasets

The following table shows the accessor code you need to add each FRED dataset to your algorithm:

Symbol	Accessor Code	Summary
CBOE		
VXGOGCLS	`Fred.CBOE.VIXOnGoogle`	CBOE Equity VIX on Google (in Index)
VXDCLS	`Fred.CBOE.VXD`	CBOE DJIA Volatility Index (in Index)
VXGSCLS	`Fred.CBOE.VIXOnGoldmanSachs`	CBOE Equity VIX on Goldman Sachs (in Index)
VXIBMCLS	`Fred.CBOE.VIXOnIBM`	CBOE Equity VIX on IBM (in Index)
VXAZNCLS	`Fred.CBOE.VIXOnAmazon`	CBOE Equity VIX on Amazon (in Index)
VXOCLS	`Fred.CBOE.VXO`	CBOE S&P 100 Volatility Index: VXO (in Index)
VXNCLS	`Fred.CBOE.VXN`	CBOE NASDAQ 100 Volatility Index (in Index)
VXTYN	`Fred.CBOE.TenYearTreasuryNoteVolatilityFutures`	CBOE 10-Year Treasury Note Volatility Futures (in Index)
RVXCLS	`Fred.CBOE.RVX`	CBOE Russell 2000 Volatility Index (in Index)
VXVCLS	`Fred.CBOE.SP500ThreeMonthVolatilityIndex`	CBOE S&P 500 3-Month Volatility Index (in Index)
VXAPLCLS	`Fred.CBOE.VIXOnApple`	CBOE Equity VIX on Apple (in Index)
VXGDXCLS	`Fred.CBOE.GoldMinersETFVolatilityIndex`	CBOE Gold Miners ETF Volatility Index (in Index)
VXFXICLS	`Fred.CBOE.ChinaETFVolatilityIndex`	CBOE China ETF Volatility Index (in Index)
VXEWZCLS	`Fred.CBOE.BrazilETFVolatilityIndex`	CBOE Brazil ETF Volatility Index (in Index)
VXEEMCLS	`Fred.CBOE.EmergingMarketsETFVolatilityIndex`	CBOE Emerging Markets ETF Volatility Index (in Index)

Symbol	Accessor Code	Summary
EVZCLS	`Fred.CBOE.EuroCurrencyETFVolatilityIndex`	CBOE EuroCurrency ETF Volatility Index (in Index)
GVZCLS	`Fred.CBOE.GoldETFVolatilityIndex`	CBOE Gold ETF Volatility Index (in Index)
OVXCLS	`Fred.CBOE.CrudeOilETFVolatilityIndex`	CBOE Crude Oil ETF Volatility Index (in Index)
VXSLVCLS	`Fred.CBOE.SilverETFVolatilityIndex`	CBOE Silver ETF Volatility Index (in Index)
VXXLECLS	`Fred.CBOE.EnergySectorETFVolatilityIndex`	CBOE Energy Sector ETF Volatility Index (in Index)
VIXCLS	`Fred.CBOE.VIX`	CBOE Volatility Index: VIX (in Index)
CentralBankInterventions		
JPINTDDMEJPY	`Fred.CentralBankInterventions.JapaneseBankPurchasesOfDmEuroAgainstJpy`	Japan Intervention: Japanese Bank purchases of DM/Euro against JPY (in 100 Million Yen)
JPINTDEXR	`Fred.CentralBankInterventions.JapaneseBankPurchasesOfUsdAgainstDm`	Japan Intervention: Japanese Bank purchases of USD against DM (in 100 Million Yen)
JPINTDUSDRP	`Fred.CentralBankInterventions.JapaneseBankPurchasesOfUsdAgainstRupiah`	Japan Intervention: Japanese Bank purchases of USD against Rupiah (in 100 Million Yen)
USINTDMRKTJPY	`Fred.CentralBankInterventions.USInterventionInMarketTransactionsInTheJpyUsd`	U.S. Intervention: in Market Transactions in the JPY/USD (Millions of USD) (in Millions of USD)
USINTDCSOTH	`Fred.CentralBankInterventions.USInterventionWithCustomerTransactionsInOtherCurrencies`	U.S. Intervention: With-Customer Transactions in Other Currencies (Millions of USD) (in Millions of USD)

Symbol	Accessor Code	Summary
USINTDCSJPY	`Fred.CentralBankInterventions.USInterventionWithCustomerTransactionsInTheJpyUsd`	U.S. Intervention: With-Customer Transactions in the JPY/USD (Millions of USD) (in Millions of USD)
USINTDCSDM	`Fred.CentralBankInterventions.USInterventionWithCustomerTransactionsInTheDemUsdEuro`	U.S. Intervention: With-Customer Transactions in the DEM/USD (Euro since 1999) (Millions of USD) (in Millions of USD)
USINTDMRKTOTH	`Fred.CentralBankInterventions.USInterventionInMarketTransactionsInOtherCurrencies`	U.S. Intervention: in Market Transactions in Other Currencies (Millions of USD) (in Millions of USD)
TRINTDEXR	`Fred.CentralBankInterventions.CentralBankOfTurkeyPurchasesOfUsd`	Turkish Intervention: Central Bank of Turkey Purchases of USD (Millions of USD) (in Millions of USD)
JPINTDUSDJPY	`Fred.CentralBankInterventions.JapaneseBankPurchasesOfUsdAgainstJpy`	Japan Intervention: Japanese Bank purchases of USD against JPY (in 100 Million Yen)
USINTDMRKTDM	`Fred.CentralBankInterventions.USInterventionInMarketTransactionsInTheDemUsdEuro`	U.S. Intervention: in Market Transactions in the DEM/USD (Euro since 1999) (Millions of USD) (in Millions of USD)
CHINTDCHFDM	`Fred.CentralBankInterventions.SwissNationalBankPurchasesOfDemAgainstChfMillionsOfDem`	Swiss Intervention: Swiss National Bank Purchases of DEM against CHF (Millions of DEM) (in Millions of DEM)
CHINTDUSDDM	`Fred.CentralBankInterventions.SwissNationalBankPurchasesOfUsdAgainstDem`	Swiss Intervention: Swiss National Bank Purchases of USD against DEM (Millions of USD) (in Millions of USD)
CHINTDUSDJPY	`Fred.CentralBankInterventions.SwissNationalBankPurchasesOfUsdAgainstJpy`	Swiss Intervention: Swiss National Bank Purchases of USD against JPY (Millions of USD) (in Millions of USD)

Symbol	Accessor Code	Summary
CHINTDCHFUSD	`Fred.CentralBankInterventions.SwissNationalBankPurchasesOfUsdAgainstChf`	Swiss Intervention: Swiss National Bank Purchases of USD against CHF (Millions of USD) (in Millions of USD)
MEXINTDUSD	`Fred.CentralBankInterventions.BancoDeMexicoPurchaseOnTheUsd`	Mexican Intervention: Banco de Mexico Purchase on the USD (in Millions of USD)
CommercialPaper		
DCPN3M	`Fred.CommercialPaper.ThreeMonthAANonfinancialCommercialPaperRate`	3-Month AA Nonfinancial Commercial Paper Rate (in Percent)
DCPN30	`Fred.CommercialPaper.OneMonthAANonfinancialCommercialPaperRate`	1-Month AA Nonfinancial Commercial Paper Rate (in Percent)
DCPN2M	`Fred.CommercialPaper.TwoMonthAANonfinancialCommercialPaperRate`	2-Month AA Nonfinancial Commercial Paper Rate (in Percent)
DCPF3M	`Fred.CommercialPaper.ThreeMonthAAFinancialCommercialPaperRate`	3-Month AA Financial Commercial Paper Rate (in Percent)
DCPF2M	`Fred.CommercialPaper.TwoMonthAAFinancialCommercialPaperRate`	2-Month AA Financial Commercial Paper Rate (in Percent)
DCPF1M	`Fred.CommercialPaper.OneMonthAAFinancialCommercialPaperRate`	1-Month AA Financial Commercial Paper Rate (in Percent)
NONFIN14A2P2VOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForA2P2Nonfinancial`	Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number)
NONFIN59A2P2VOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForA2P2Nonfinancial`	Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number)

Symbol	Accessor Code	Summary
NONFIN59A2P2AMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForA2P2Nonfinancial`	Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars)
NONFIN4180AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForAANonfinancial`	Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number)
ABGT80AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForAAAssetBacked`	Total Value of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars)
NONFIN4180AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForAANonfinancial`	Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars)
NONFIN4180A2P2VOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForA2P2Nonfinancial`	Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number)
NONFIN4180A2P2AMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForA2P2Nonfinancial`	Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars)
NONFIN2140AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForAANonfinancial`	Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number)

Symbol	Accessor Code	Summary
NONFIN2140AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForAANonfinancial`	Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars)
NONFIN2140A2P2VOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForA2P2Nonfinancial`	Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number)
NONFIN2140A2P2AMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForA2P2Nonfinancial`	Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars)
NONFIN14AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForAANonfinancial`	Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number)
NONFIN1020A2P2VOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForA2P2Nonfinancial`	Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number)
NONFIN1020AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForAANonfinancial`	Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars)
AB2140AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForAAAssetBacked`	Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars)

Symbol	Accessor Code	Summary
NONFIN1020AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForAANonfinancial`	Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number)
NONFIN14A2P2AMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForA2P2Nonfinancial`	Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars)
NONFIN14AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForAANonfinancial`	Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars)
MKT14MKTAMT	`Fred.CommercialPaper.TotalValueofCommercialPaperIssueswithaMaturityBetween1and4Days`	Total Value of Commercial Paper Issues with a Maturity Between 1 and 4 Days (in Millions of Dollars)
NONFIN1020A2P2AMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForA2P2Nonfinancial`	Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars)
FINGT80AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForAAFinancial`	Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number)
FIN1020AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForAAFinancial`	Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number)
FIN14AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForAAFinancial`	Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars)

Symbol	Accessor Code	Summary
FIN14AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForAAFinancial`	Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number)
MKT1020MKTAMT	`Fred.CommercialPaper.TotalValueofCommercialPaperIssueswithaMaturityBetween10And20Days`	Total Value of Commercial Paper Issues with a Maturity Between 10 and 20 Days (in Millions of Dollars)
MKT1020MKTVOL	`Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween10And20Days`	Number of Commercial Paper Issues with a Maturity Between 10 and 20 Days (in Number)
FIN2140AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForAAFinancial`	Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars)
MKT14MKTVOL	`Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween1and4Days`	Number of Commercial Paper Issues with a Maturity Between 1 and 4 Days (in Number)
MKT2140MKTAMT	`Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityBetween21and40Days`	Total Value of Issuers of Commercial Paper with a Maturity Between 21 and 40 Days (in Millions of Dollars)
MKT2140MKTVOL	`Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween21and40Days`	Number of Commercial Paper Issues with a Maturity Between 21 and 40 Days (in Number)
FIN2140AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForAAFinancial`	Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number)
MKT4180MKTAMT	`Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityBetween41and80Days`	Total Value of Issuers of Commercial Paper with a Maturity Between 41 and 80 Days (in Millions of Dollars)

Symbol	Accessor Code	Summary
NONFIN59AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForAANonfinancial`	Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars)
MKT4180MKTVOL	`Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween41and80Days`	Number of Commercial Paper Issues with a Maturity Between 41 and 80 Days (in Number)
MKT59MKTVOL	`Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween5and9Days`	Number of Commercial Paper Issues with a Maturity Between 5 and 9 Days (in Number)
MKTGT80MKTAMT	`Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityGreaterThan80Days`	Total Value of Issuers of Commercial Paper with a Maturity Greater Than 80 Days (in Millions of Dollars)
MKTGT80MKTVOL	`Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityGreaterThan80Days`	Number of Commercial Paper Issues with a Maturity Greater Than 80 Days (in Number)
FIN4180AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForAAFinancial`	Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars)
FIN4180AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForAAFinancial`	Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number)
AB4180AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForAAAssetBacked`	Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars)

Symbol	Accessor Code	Summary
FIN59AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForAAFinancial`	Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars)
FIN59AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForAAFinancial`	Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number)
FINGT80AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForAAFinancial`	Total Value of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars)
FIN1020AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForAAFinancial`	Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars)
AB2140AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForAAAssetBacked`	Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number)
MKT59MKTAMT	`Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityBetween5and9Days`	Total Value of Issuers of Commercial Paper with a Maturity Between 5 and 9 Days (in Millions of Dollars)
ABGT80AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForAAAssetBacked`	Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number)
NONFIN59AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForAANonfinancial`	Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number)

Symbol	Accessor Code	Summary
RIFSPPAAAD15NB	`Fred.CommercialPaper.FifteenDayAAAssetbackedCommercialPaperInterestRate`	15-Day AA Asset-backed Commercial Paper Interest Rate (in Percent)
AB59AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForAAAssetBacked`	Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars)
AB4180AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForAAAssetBacked`	Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number)
RIFSPPNA2P2D15NB	`Fred.CommercialPaper.FifteenDayA2P2NonfinancialCommercialPaperInterestRate`	15-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPNA2P2D07NB	`Fred.CommercialPaper.SevenDayA2P2NonfinancialCommercialPaperInterestRate`	7-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPNA2P2D01NB	`Fred.CommercialPaper.OvernightA2P2NonfinancialCommercialPaperInterestRate`	Overnight A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPFAAD90NB	`Fred.CommercialPaper.NinetyDayAAFinancialCommercialPaperInterestRate`	90-Day AA Financial Commercial Paper Interest Rate (in Percent)
RIFSPPAAAD01NB	`Fred.CommercialPaper.OvernightAAAssetbackedCommercialPaperInterestRate`	Overnight AA Asset-backed Commercial Paper Interest Rate (in Percent)
RIFSPPNA2P2D30NB	`Fred.CommercialPaper.Three0DayA2P2NonfinancialCommercialPaperInterestRate`	30-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPFAAD60NB	`Fred.CommercialPaper.SixtyDayAAFinancialCommercialPaperInterestRate`	60-Day AA Financial Commercial Paper Interest Rate (in Percent)

Symbol	Accessor Code	Summary
RIFSPPFAAD30NB	`Fred.CommercialPaper.Three0DayAAFinancialCommercialPaperInterestRate`	30-Day AA Financial Commercial Paper Interest Rate (in Percent)
NONFINGT80A2P2AMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForA2P2Nonfinancial`	Total Value of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars)
RIFSPPAAAD30NB	`Fred.CommercialPaper.Three0DayAAAssetbackedCommercialPaperInterestRate`	30-Day AA Asset-backed Commercial Paper Interest Rate (in Percent)
RIFSPPAAAD60NB	`Fred.CommercialPaper.SixtyDayAAAssetbackedCommercialPaperInterestRate`	60-Day AA Asset-backed Commercial Paper Interest Rate (in Percent)
RIFSPPAAAD90NB	`Fred.CommercialPaper.NinetyDayAAAssetbackedCommercialPaperInterestRate`	90-Day AA Asset-backed Commercial Paper Interest Rate (in Percent)
RIFSPPFAAD15NB	`Fred.CommercialPaper.FifteenDayAAFinancialCommercialPaperInterestRate`	15-Day AA Financial Commercial Paper Interest Rate (in Percent)
RIFSPPFAAD07NB	`Fred.CommercialPaper.SevenDayAAFinancialCommercialPaperInterestRate`	7-Day AA Financial Commercial Paper Interest Rate (in Percent)
RIFSPPAAAD07NB	`Fred.CommercialPaper.SevenDayAAAssetbackedCommercialPaperInterestRate`	7-Day AA Asset-backed Commercial Paper Interest Rate (in Percent)
RIFSPPFAAD01NB	`Fred.CommercialPaper.OvernightAAFinancialCommercialPaperInterestRate`	Overnight AA Financial Commercial Paper Interest Rate (in Percent)

Symbol	Accessor Code	Summary
RIFSPPNA2P2D60NB	`Fred.CommercialPaper.SixtyDayA2P2NonfinancialCommercialPaperInterestRate`	60-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent)
AB59AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForAAAssetBacked`	Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number)
AB14AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForAAAssetBacked`	Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number)
NONFINGT80A2P2VOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForA2P2Nonfinancial`	Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number)
AB14AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForAAAssetBacked`	Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars)
RIFSPPNA2P2D90NB	`Fred.CommercialPaper.NinetyDayA2P2NonfinancialCommercialPaperInterestRate`	90-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent)
AB1020AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForAAAssetBacked`	Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number)
NONFINGT80AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForAANonfinancial`	Total Value of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars)

Symbol	Accessor Code	Summary
RIFSPPNAAD01NB	`Fred.CommercialPaper.OvernightAANonfinancialCommercialPaperInterestRate`	Overnight AA Nonfinancial Commercial Paper Interest Rate (in Percent)
AB1020AAAMT	`Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForAAAssetBacked`	Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars)
RIFSPPNAAD07NB	`Fred.CommercialPaper.SevenDayAANonfinancialCommercialPaperInterestRate`	7-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPNAAD90NB	`Fred.CommercialPaper.NinetyDayAANonfinancialCommercialPaperInterestRate`	90-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPNAAD15NB	`Fred.CommercialPaper.FifteenDayAANonfinancialCommercialPaperInterestRate`	15-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPNAAD30NB	`Fred.CommercialPaper.Three0DayAANonfinancialCommercialPaperInterestRate`	30-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent)
RIFSPPNAAD60NB	`Fred.CommercialPaper.SixtyDayAANonfinancialCommercialPaperInterestRate`	60-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent)
NONFINGT80AAVOL	`Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForAANonfinancial`	Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number)
CPFF	`Fred.CommercialPaper.ThreeMonthCommercialPaperMinusFederalFundsRate`	3-Month Commercial Paper Minus Federal Funds Rate (in Percent)
ICEBofAML		

Symbol	Accessor Code	Summary
BAMLEM1BRRAAA2ACRPITRIV	`Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM1RAAA2ALCRPIUSTRIV	`Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMRACRPIASIATRIV	`Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMALLCRPIASIAUSTRIV	`Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM4BRRBLCRPITRIV	`Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM4RBLLCRPIUSTRIV	`Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM3BRRBBCRPITRIV	`Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM3RBBLCRPIUSTRIV	`Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM2BRRBBBCRPITRIV	`Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)

Symbol	Accessor Code	Summary
BAMLEM2RBBBLCRPIUSTRIV	`Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEM5BCOCRPITRIV	`Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMXOCOLCRPIUSTRIV	`Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMCBPITRIV	`Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexTotalReturnIndexValue`	ICE BofAML Emerging Markets Corporate Plus Index Total Return Index Value (in Index)
BAMLEMEBCRPIETRIV	`Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMRECRPIEMEATRIV	`Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMELLCRPIEMEAUSTRIV	`Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMFSFCRPITRIV	`Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMFLFLCRPIUSTRIV	`Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)

Symbol	Accessor Code	Summary
BAMLEMIBHGCRPITRIV	`Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMHGHGLCRPIUSTRIV	`Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMHBHYCRPITRIV	`Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMHYHYLCRPIUSTRIV	`Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMRLCRPILATRIV	`Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMLLLCRPILAUSTRIV	`Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMNSNFCRPITRIV	`Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMNFNFLCRPIUSTRIV	`Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLC0A0CM	`Fred.ICEBofAML.USCorporateMasterOptionAdjustedSpread`	ICE BofAML US Corporate Master Option-Adjusted Spread (in Percent)

Symbol	Accessor Code	Summary
BAMLH0A0HYM2	`Fred.ICEBofAML.USHighYieldMasterIIOptionAdjustedSpread`	ICE BofAML US High Yield Master II Option-Adjusted Spread (in Percent)
BAMLC1A0C13Y	`Fred.ICEBofAML.USCorporate1To3YearOptionAdjustedSpread`	ICE BofAML US Corporate 1-3 Year Option-Adjusted Spread (in Percent)
BAMLC7A0C1015Y	`Fred.ICEBofAML.USCorporate10To15YearOptionAdjustedSpread`	ICE BofAML US Corporate 10-15 Year Option-Adjusted Spread (in Percent)
BAMLC8A0C15PY	`Fred.ICEBofAML.USCorporateMoreThan15YearOptionAdjustedSpread`	ICE BofAML US Corporate 15+ Year Option-Adjusted Spread (in Percent)
BAMLC2A0C35Y	`Fred.ICEBofAML.USCorporate3To5YearOptionAdjustedSpread`	ICE BofAML US Corporate 3-5 Year Option-Adjusted Spread (in Percent)
BAMLC3A0C57Y	`Fred.ICEBofAML.USCorporate5To7YearOptionAdjustedSpread`	ICE BofAML US Corporate 5-7 Year Option-Adjusted Spread (in Percent)
BAMLC4A0C710Y	`Fred.ICEBofAML.USCorporate7To10YearOptionAdjustedSpread`	ICE BofAML US Corporate 7-10 Year Option-Adjusted Spread (in Percent)
BAMLEMPUPUBSLCRPIUSTRIV	`Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMUBCRPIUSTRIV	`Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue`	ICE BofAML US Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index)
BAMLEMCLLCRPIUSTRIV	`Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexTotalReturnIndexValue`	ICE BofAML US Emerging Markets Liquid Corporate Plus Index Total Return Index Value (in Index)
BAMLHE00EHYITRIV	`Fred.ICEBofAML.EuroHighYieldIndexTotalReturnIndexValue`	ICE BofAML Euro High Yield Index Total Return Index Value (in Index)

Symbol	Accessor Code	Summary
BAMLCC1A013YTRIV	Fred.ICEBofAML.USCorp1To3YearsTotalReturnIndexValue	ICE BofAML US Corp 1-3yr Total Return Index Value (in Index)
BAMLCC7A01015YTRIV	Fred.ICEBofAML.USCorp10To15TotalReturnIndexValue	ICE BofAML US Corp 10-15yr Total Return Index Value (in Index)
BAMLCC8A015PYTRIV	Fred.ICEBofAML.USCorpMoreThan15YearsTotalReturnIndexValue	ICE BofAML US Corp 15+yr Total Return Index Value (in Index)
BAMLCC2A035YTRIV	Fred.ICEBofAML.USCorpeTo5YearsTotalReturnIndexValue	ICE BofAML US Corp 3-5yr Total Return Index Value (in Index)
BAMLCC3A057YTRIV	Fred.ICEBofAML.USCorp5To7YearsTotalReturnIndexValue	ICE BofAML US Corp 5-7yr Total Return Index Value (in Index)
BAMLCC4A0710YTRIV	Fred.ICEBofAML.USCorporate7To10YearsTotalReturnIndexValue	ICE BofAML US Corporate 7-10yr Total Return Index Value (in Index)
BAMLCC0A3ATRIV	Fred.ICEBofAML.USCorpATotalReturnIndexValue	ICE BofAML US Corp A Total Return Index Value (in Index)
BAMLCC0A2AATRIV	Fred.ICEBofAML.USCorpAATotalReturnIndexValue	ICE BofAML US Corp AA Total Return Index Value (in Index)
BAMLCC0A1AAATRIV	Fred.ICEBofAML.USCorpAAATotalReturnIndexValue	ICE BofAML US Corp AAA Total Return Index Value (in Index)
BAMLHYH0A2BTRIV	Fred.ICEBofAML.USHighYieldBTotalReturnIndexValue	ICE BofAML US High Yield B Total Return Index Value (in Index)
BAMLHYH0A1BBTRIV	Fred.ICEBofAML.USHighYieldBBTotalReturnIndexValue	ICE BofAML US High Yield BB Total Return Index Value (in Index)
BAMLCC0A4BBBTRIV	Fred.ICEBofAML.USCorpBBBTotalReturnIndexValue	ICE BofAML US Corp BBB Total Return Index Value (in Index)
BAMLHYH0A3CMTRIV	Fred.ICEBofAML.USHighYieldCCCorBelowTotalReturnIndexValue	ICE BofAML US High Yield CCC or Below Total Return Index Value (in Index)
BAMLCC0A0CMTRIV	Fred.ICEBofAML.USCorpMasterTotalReturnIndexValue	ICE BofAML US Corp Master Total Return Index Value (in Index)

Symbol	Accessor Code	Summary
BAMLHYH0A0HYM2TRIV	`Fred.ICEBofAML.USHighYieldMasterIITotalReturnIndexValue`	ICE BofAML US High Yield Master II Total Return Index Value (in Index)
BAMLEM1BRRAAA2ACRPIOAS	`Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM1RAAA2ALCRPIUSOAS	`Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMRACRPIASIAOAS	`Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMALLCRPIASIAUSOAS	`Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM4BRRBLCRPIOAS	`Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM4RBLLCRPIUSOAS	`Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM3BRRBBCRPIOAS	`Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM3RBBLCRPIUSOAS	`Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)

Symbol	Accessor Code	Summary
BAMLEM2BRRBBBCRPIOAS	`Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM2RBBBLCRPIUSOAS	`Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEM5BCOCRPIOAS	`Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMXOCOLCRPIUSOAS	`Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMCBPIOAS	`Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexOptionAdjustedSpread`	ICE BofAML Emerging Markets Corporate Plus Index Option-Adjusted Spread (in Percent)
BAMLEMEBCRPIEOAS	`Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMRECRPIEMEAOAS	`Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMELLCRPIEMEAUSOAS	`Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMFSFCRPIOAS	`Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)

Symbol	Accessor Code	Summary
BAMLEMFLFLCRPIUSOAS	`Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMIBHGCRPIOAS	`Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMHGHGLCRPIUSOAS	`Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMHBHYCRPIOAS	`Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMHYHYLCRPIUSOAS	`Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMRLCRPILAOAS	`Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMLLLCRPILAUSOAS	`Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMNSNFCRPIOAS	`Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)

Symbol	Accessor Code	Summary
BAMLEMNFNFLCRPIUSOAS	`Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMPUPUBSLCRPIUSOAS	`Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMUBCRPIUSOAS	`Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread`	ICE BofAML US Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent)
BAMLEMCLLCRPIUSOAS	`Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexOptionAdjustedSpread`	ICE BofAML US Emerging Markets Liquid Corporate Plus Index Option-Adjusted Spread (in Percent)
BAMLHE00EHYIOAS	`Fred.ICEBofAML.EuroHighYieldIndexOptionAdjustedSpread`	ICE BofAML Euro High Yield Index Option-Adjusted Spread (in Percent)
BAMLC0A3CA	`Fred.ICEBofAML.USCorporateAOptionAdjustedSpread`	ICE BofAML US Corporate A Option-Adjusted Spread (in Percent)
BAMLC0A2CAA	`Fred.ICEBofAML.USCorporateAAOptionAdjustedSpread`	ICE BofAML US Corporate AA Option-Adjusted Spread (in Percent)
BAMLC0A1CAAA	`Fred.ICEBofAML.USCorporateAAAOptionAdjustedSpread`	ICE BofAML US Corporate AAA Option-Adjusted Spread (in Percent)
BAMLH0A2HYB	`Fred.ICEBofAML.USHighYieldBOptionAdjustedSpread`	ICE BofAML US High Yield B Option-Adjusted Spread (in Percent)
BAMLH0A1HYBB	`Fred.ICEBofAML.USHighYieldBBOptionAdjustedSpread`	ICE BofAML US High Yield BB Option-Adjusted Spread (in Percent)

Symbol	Accessor Code	Summary
BAMLC0A4CBBB	`Fred.ICEBofAML.USCorporateBBBOptionAdjustedSpread`	ICE BofAML US Corporate BBB Option-Adjusted Spread (in Percent)
BAMLH0A3HYC	`Fred.ICEBofAML.USHighYieldCCCorBelowOptionAdjustedSpread`	ICE BofAML US High Yield CCC or Below Option-Adjusted Spread (in Percent)
BAMLEM1BRRAAA2ACRPIEY	`Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexEffectiveYield`	ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM1RAAA2ALCRPIUSEY	`Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield`	ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMRACRPIASIAEY	`Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexEffectiveYield`	ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMALLCRPIASIAUSEY	`Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield`	ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM4BRRBLCRPIEY	`Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexEffectiveYield`	ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM4RBLLCRPIUSEY	`Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield`	ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM3BRRBBCRPIEY	`Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexEffectiveYield`	ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM3RBBLCRPIUSEY	`Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield`	ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)

Symbol	Accessor Code	Summary
BAMLEM2BRRBBBCRPIEY	Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM2RBBBLCRPIUSEY	Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEM5BCOCRPIEY	Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMXOCOLCRPIUSEY	Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMCBPIEY	Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexEffectiveYield	ICE BofAML Emerging Markets Corporate Plus Index Effective Yield (in Percent)
BAMLEMEBCRPIEEY	Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLHE00EHYIEY	Fred.ICEBofAML.EuroHighYieldIndexEffectiveYield	ICE BofAML Euro High Yield Index Effective Yield (in Percent)
BAMLEMRECRPIEMEAEY	Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMELLCRPIEMEAUSEY	Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMFSFCRPIEY	Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)

Symbol	Accessor Code	Summary
BAMLEMFLFLCRPIUSEY	Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMIBHGCRPIEY	Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMHGHGLCRPIUSEY	Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMHBHYCRPIEY	Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMHYHYLCRPIUSEY	Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMRLCRPILAEY	Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMLLLCRPILAUSEY	Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMNSNFCRPIEY	Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexEffectiveYield	ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)

Symbol	Accessor Code	Summary
BAMLEMNFNFLCRPIUSEY	Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMPUPUBSLCRPIUSEY	Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield	ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLC1A0C13YEY	Fred.ICEBofAML.USCorporate1ThreeYearEffectiveYield	ICE BofAML US Corporate 1-3 Year Effective Yield (in Percent)
BAMLC7A0C1015YEY	Fred.ICEBofAML.USCorporate10To15YearEffectiveYield	ICE BofAML US Corporate 10-15 Year Effective Yield (in Percent)
BAMLC8A0C15PYEY	Fred.ICEBofAML.USCorporateMoreThan15YearEffectiveYield	ICE BofAML US Corporate 15+ Year Effective Yield (in Percent)
BAMLC2A0C35YEY	Fred.ICEBofAML.USCorporate3To5YearEffectiveYield	ICE BofAML US Corporate 3-5 Year Effective Yield (in Percent)
BAMLC3A0C57YEY	Fred.ICEBofAML.USCorporate5To7YearEffectiveYield	ICE BofAML US Corporate 5-7 Year Effective Yield (in Percent)
BAMLC4A0C710YEY	Fred.ICEBofAML.USCorporate7To10YearEffectiveYield	ICE BofAML US Corporate 7-10 Year Effective Yield (in Percent)
BAMLC0A3CAEY	Fred.ICEBofAML.USCorporateAEffectiveYield	ICE BofAML US Corporate A Effective Yield (in Percent)
BAMLC0A2CAAEY	Fred.ICEBofAML.USCorporateAAEffectiveYield	ICE BofAML US Corporate AA Effective Yield (in Percent)
BAMLC0A1CAAAEY	Fred.ICEBofAML.USCorporateAAAEffectiveYield	ICE BofAML US Corporate AAA Effective Yield (in Percent)
BAMLH0A2HYBEY	Fred.ICEBofAML.USHighYieldBEffectiveYield	ICE BofAML US High Yield B Effective Yield (in Percent)
BAMLH0A1HYBBEY	Fred.ICEBofAML.USHighYieldBBEffectiveYield	ICE BofAML US High Yield BB Effective Yield (in Percent)

Symbol	Accessor Code	Summary
BAMLC0A4CBBBEY	`Fred.ICEBofAML.USCorporateBBBEffectiveYield`	ICE BofAML US Corporate BBB Effective Yield (in Percent)
BAMLH0A3HYCEY	`Fred.ICEBofAML.USHighYieldCCCorBelowEffectiveYield`	ICE BofAML US High Yield CCC or Below Effective Yield (in Percent)
BAMLC0A0CMEY	`Fred.ICEBofAML.USCorporateMasterEffectiveYield`	ICE BofAML US Corporate Master Effective Yield (in Percent)
BAMLEMUBCRPIUSEY	`Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexEffectiveYield`	ICE BofAML US Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent)
BAMLEMCLLCRPIUSEY	`Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexEffectiveYield`	ICE BofAML US Emerging Markets Liquid Corporate Plus Index Effective Yield (in Percent)
BAMLH0A0HYM2EY	`Fred.ICEBofAML.USHighYieldMasterIIEffectiveYield`	ICE BofAML US High Yield Master II Effective Yield (in Percent)
BAMLEM1BRRAAA2ACRPISYTW	`Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM1RAAA2ALCRPIUSSYTW	`Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMRACRPIASIASYTW	`Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMALLCRPIASIAUSSYTW	`Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)

Symbol	Accessor Code	Summary
BAMLEM4BRRBLCRPISYTW	Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM4RBLLCRPIUSSYTW	Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM3BRRBBCRPISYTW	Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM3RBBLCRPIUSSYTW	Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM2BRRBBBCRPISYTW	Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM2RBBBLCRPIUSSYTW	Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEM5BCOCRPISYTW	Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMXOCOLCRPIUSSYTW	Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMCBPISYTW	Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexSemiAnnualYieldtoWorst	ICE BofAML Emerging Markets Corporate Plus Index Semi-Annual Yield to Worst (in Percent)

Symbol	Accessor Code	Summary
BAMLEMEBCRPIESYTW	Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLHE00EHYISYTW	Fred.ICEBofAML.EuroHighYieldIndexSemiAnnualYieldtoWorst	ICE BofAML Euro High Yield Index Semi-Annual Yield to Worst (in Percent)
BAMLEMRECRPIEMEASYTW	Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMELLCRPIEMEAUSSYTW	Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMFSFCRPISYTW	Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMFLFLCRPIUSSYTW	Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMIBHGCRPISYTW	Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMHGHGLCRPIUSSYTW	Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMHBHYCRPISYTW	Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)

Symbol	Accessor Code	Summary
BAMLEMHYHYLCRPIUSSYTW	`Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMRLCRPILASYTW	`Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMLLLCRPILAUSSYTW	`Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMNSNFCRPISYTW	`Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMNFNFLCRPIUSSYTW	`Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMPTPRVICRPISYTW	`Fred.ICEBofAML.PrivateSectorIssuersEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Private Sector Issuers Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMPVPRIVSLCRPIUSSYTW	`Fred.ICEBofAML.PrivateSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Private Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMPBPUBSICRPISYTW	`Fred.ICEBofAML.PublicSectorIssuersEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML Public Sector Issuers Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)

Symbol	Accessor Code	Summary
BAMLEMPUPUBSLCRPIUSSYTW	Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst	ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLC1A0C13YSYTW	Fred.ICEBofAML.USCorporate1To3YearSemiAnnualYieldtoWorst	ICE BofAML US Corporate 1-3 Year Semi-Annual Yield to Worst (in Percent)
BAMLC7A0C1015YSYTW	Fred.ICEBofAML.USCorporate10To15YearSemiAnnualYieldtoWorst	ICE BofAML US Corporate 10-15 Year Semi-Annual Yield to Worst (in Percent)
BAMLC8A0C15PYSYTW	Fred.ICEBofAML.USCorporateMoreThan15YearSemiAnnualYieldtoWorst	ICE BofAML US Corporate 15+ Year Semi-Annual Yield to Worst (in Percent)
BAMLC2A0C35YSYTW	Fred.ICEBofAML.USCorporate3To5YearSemiAnnualYieldtoWorst	ICE BofAML US Corporate 3-5 Year Semi-Annual Yield to Worst (in Percent)
BAMLC3A0C57YSYTW	Fred.ICEBofAML.USCorporate5To7YearSemiAnnualYieldtoWorst	ICE BofAML US Corporate 5-7 Year Semi-Annual Yield to Worst (in Percent)
BAMLC4A0C710YSYTW	Fred.ICEBofAML.USCorporate7To10YearSemiAnnualYieldtoWorst	ICE BofAML US Corporate 7-10 Year Semi-Annual Yield to Worst (in Percent)
BAMLC0A3CASYTW	Fred.ICEBofAML.USCorporateASemiAnnualYieldtoWorst	ICE BofAML US Corporate A Semi-Annual Yield to Worst (in Percent)
BAMLC0A2CAASYTW	Fred.ICEBofAML.USCorporateAASemiAnnualYieldtoWorst	ICE BofAML US Corporate AA Semi-Annual Yield to Worst (in Percent)
BAMLC0A1CAAASYTW	Fred.ICEBofAML.USCorporateAAASemiAnnualYieldtoWorst	ICE BofAML US Corporate AAA Semi-Annual Yield to Worst (in Percent)
BAMLH0A2HYBSYTW	Fred.ICEBofAML.USHighYieldBSemiAnnualYieldtoWorst	ICE BofAML US High Yield B Semi-Annual Yield to Worst (in Percent)

Symbol	Accessor Code	Summary
BAMLH0A1HYBBSYTW	`Fred.ICEBofAML.USHighYieldBBSemiAnnualYieldtoWorst`	ICE BofAML US High Yield BB Semi-Annual Yield to Worst (in Percent)
BAMLC0A4CBBBSYTW	`Fred.ICEBofAML.USCorporateBBBSemiAnnualYieldtoWorst`	ICE BofAML US Corporate BBB Semi-Annual Yield to Worst (in Percent)
BAMLH0A3HYCSYTW	`Fred.ICEBofAML.USHighYieldCCCorBelowSemiAnnualYieldtoWorst`	ICE BofAML US High Yield CCC or Below Semi-Annual Yield to Worst (in Percent)
BAMLC0A0CMSYTW	`Fred.ICEBofAML.USCorporateMasterSemiAnnualYieldtoWorst`	ICE BofAML US Corporate Master Semi-Annual Yield to Worst (in Percent)
BAMLEMUBCRPIUSSYTW	`Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst`	ICE BofAML US Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent)
BAMLEMCLLCRPIUSSYTW	`Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexSemiAnnualYieldtoWorst`	ICE BofAML US Emerging Markets Liquid Corporate Plus Index Semi-Annual Yield to Worst (in Percent)
BAMLH0A0HYM2SYTW	`Fred.ICEBofAML.USHighYieldMasterIISemiAnnualYieldtoWorst`	ICE BofAML US High Yield Master II Semi-Annual Yield to Worst (in Percent)
LIBOR		
CHFONTD156N	`Fred.LIBOR.SpotNextBasedOnSwissFranc`	Spot Next London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)
JPYONTD156N	`Fred.LIBOR.SpotNextBasedOnJapaneseYen`	Spot Next London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)
JPY6MTD156N	`Fred.LIBOR.SixMonthBasedOnJapaneseYen`	6-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)
JPY3MTD156N	`Fred.LIBOR.ThreeMonthBasedOnJapaneseYen`	3-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)

Symbol	Accessor Code	Summary
USD6MTD156N	`Fred.LIBOR.SixMonthBasedOnUSD`	6-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
JPY1MTD156N	`Fred.LIBOR.OneMonthBasedOnJapaneseYen`	1-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)
JPY12MD156N	`Fred.LIBOR.TwelveMonthBasedOnJapaneseYen`	12-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)
GBP12MD156N	`Fred.LIBOR.TwelveMonthBasedOnBritishPound`	12-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
GBP1MTD156N	`Fred.LIBOR.OneMonthBasedOnBritishPound`	1-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
GBP1WKD156N	`Fred.LIBOR.OneWeekBasedOnBritishPound`	1-Week London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
GBP2MTD156N	`Fred.LIBOR.TwoMonthBasedOnBritishPound`	2-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
GBP3MTD156N	`Fred.LIBOR.ThreeMonthBasedOnBritishPound`	3-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
JPY1WKD156N	`Fred.LIBOR.OneWeekBasedOnJapaneseYen`	1-Week London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)
JPY2MTD156N	`Fred.LIBOR.TwoMonthBasedOnJapaneseYen`	2-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent)
CHF6MTD156N	`Fred.LIBOR.SixMonthBasedOnSwissFranc`	6-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)
CHF3MTD156N	`Fred.LIBOR.ThreeMonthBasedOnSwissFranc`	3-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)

Symbol	Accessor Code	Summary
USD1MTD156N	`Fred.LIBOR.OneMonthBasedOnUSD`	1-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
CHF12MD156N	`Fred.LIBOR.TwelveMonthBasedOnSwissFranc`	12-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)
USD12MD156N	`Fred.LIBOR.TwelveMonthBasedOnUSD`	12-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
CHF1MTD156N	`Fred.LIBOR.OneMonthBasedOnSwissFranc`	1-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)
CHF1WKD156N	`Fred.LIBOR.OneWeekBasedOnSwissFranc`	1-Week London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)
CHF2MTD156N	`Fred.LIBOR.TwoMonthBasedOnSwissFranc`	2-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent)
EUR12MD156N	`Fred.LIBOR.TwelveMonthBasedOnEuro`	12-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent)
GBP6MTD156N	`Fred.LIBOR.SixMonthBasedOnBritishPound`	6-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
EUR1MTD156N	`Fred.LIBOR.OneMonthBasedOnEuro`	1-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent)
EUR2MTD156N	`Fred.LIBOR.TwoMonthBasedOnEuro`	2-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent)
EUR3MTD156N	`Fred.LIBOR.ThreeMonthBasedOnEuro`	3-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent)
EUR6MTD156N	`Fred.LIBOR.SixMonthBasedOnEuro`	6-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent)

Symbol	Accessor Code	Summary
EURONTD156N	`Fred.LIBOR.OvernightBasedOnEuro`	Overnight London Interbank Offered Rate (LIBOR), based on Euro (in Percent)
USD1WKD156N	`Fred.LIBOR.OneWeekBasedOnUSD`	1-Week London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
USD2MTD156N	`Fred.LIBOR.TwoMonthBasedOnUSD`	2-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
USD3MTD156N	`Fred.LIBOR.ThreeMonthBasedOnUSD`	3-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
USDONTD156N	`Fred.LIBOR.OvernightBasedOnUSD`	Overnight London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent)
EUR1WKD156N	`Fred.LIBOR.OneWeekBasedOnEuro`	1-Week London Interbank Offered Rate (LIBOR), based on Euro (in Percent)
GBPONTD156N	`Fred.LIBOR.OvernightBasedOnBritishPound`	Overnight London Interbank Offered Rate (LIBOR), based on British Pound (in Percent)
OECDRecessionIndicators		
4BIGEURORECDM	`Fred.OECDRecessionIndicators.FourBigEuropeanCountriesFromPeakThroughTheTrough`	OECD based Recession Indicators for Four Big European Countries from the Peak through the Trough (in +1 or 0)
AUSRECDM	`Fred.OECDRecessionIndicators.AustraliaFromPeakThroughTheTrough`	OECD based Recession Indicators for Australia from the Peak through the Trough (in +1 or 0)
AUTRECDM	`Fred.OECDRecessionIndicators.AustriaFromPeakThroughTheTrough`	OECD based Recession Indicators for Austria from the Peak through the Trough (in +1 or 0)
BELRECDM	`Fred.OECDRecessionIndicators.BelgiumFromPeakThroughTheTrough`	OECD based Recession Indicators for Belgium from the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
BRARECDM	`Fred.OECDRecessionIndicators.BrazilFromPeakThroughTheTrough`	OECD based Recession Indicators for Brazil from the Peak through the Trough (in +1 or 0)
CANRECDM	`Fred.OECDRecessionIndicators.CanadaFromPeakThroughTheTrough`	OECD based Recession Indicators for Canada from the Peak through the Trough (in +1 or 0)
CHERECDM	`Fred.OECDRecessionIndicators.SwitzerlandFromPeakThroughTheTrough`	OECD based Recession Indicators for Switzerland from the Peak through the Trough (in +1 or 0)
CHLRECDM	`Fred.OECDRecessionIndicators.ChileFromPeakThroughTheTrough`	OECD based Recession Indicators for Chile from the Peak through the Trough (in +1 or 0)
CHNRECDM	`Fred.OECDRecessionIndicators.ChinaFromPeakThroughTheTrough`	OECD based Recession Indicators for China from the Peak through the Trough (in +1 or 0)
CZERECDM	`Fred.OECDRecessionIndicators.CzechRepublicFromPeakThroughTheTrough`	OECD based Recession Indicators for the Czech Republic from the Peak through the Trough (in +1 or 0)
DEURECDM	`Fred.OECDRecessionIndicators.GermanyFromPeakThroughTheTrough`	OECD based Recession Indicators for Germany from the Peak through the Trough (in +1 or 0)
DNKRECDM	`Fred.OECDRecessionIndicators.DenmarkFromPeakThroughTheTrough`	OECD based Recession Indicators for Denmark from the Peak through the Trough (in +1 or 0)
ESPRECDM	`Fred.OECDRecessionIndicators.SpainFromPeakThroughTheTrough`	OECD based Recession Indicators for Spain from the Peak through the Trough (in +1 or 0)
ESTRECDM	`Fred.OECDRecessionIndicators.EstoniaFromPeakThroughTheTrough`	OECD based Recession Indicators for Estonia from the Peak through the Trough (in +1 or 0)
EURORECDM	`Fred.OECDRecessionIndicators.EuroAreaFromPeakThroughTheTrough`	OECD based Recession Indicators for Euro Area from the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
FINRECDM	`Fred.OECDRecessionIndicators.FinlandFromPeakThroughTheTrough`	OECD based Recession Indicators for Finland from the Peak through the Trough (in +1 or 0)
FRARECDM	`Fred.OECDRecessionIndicators.FranceFromPeakThroughTheTrough`	OECD based Recession Indicators for France from the Peak through the Trough (in +1 or 0)
GBRRECDM	`Fred.OECDRecessionIndicators.UnitedKingdomFromPeakThroughTheTrough`	OECD based Recession Indicators for the United Kingdom from the Peak through the Trough (in +1 or 0)
GRCRECDM	`Fred.OECDRecessionIndicators.GreeceFromPeakThroughTheTrough`	OECD based Recession Indicators for Greece from the Peak through the Trough (in +1 or 0)
HUNRECDM	`Fred.OECDRecessionIndicators.HungaryFromPeakThroughTheTrough`	OECD based Recession Indicators for Hungary from the Peak through the Trough (in +1 or 0)
IDNRECDM	`Fred.OECDRecessionIndicators.IndonesiaFromPeakThroughTheTrough`	OECD based Recession Indicators for Indonesia from the Peak through the Trough (in +1 or 0)
INDRECDM	`Fred.OECDRecessionIndicators.IndiaFromPeakThroughTheTrough`	OECD based Recession Indicators for India from the Peak through the Trough (in +1 or 0)
IRLRECDM	`Fred.OECDRecessionIndicators.IrelandFromPeakThroughTheTrough`	OECD based Recession Indicators for Ireland from the Peak through the Trough (in +1 or 0)
ISRRECDM	`Fred.OECDRecessionIndicators.IsraelFromPeakThroughTheTrough`	OECD based Recession Indicators for Israel from the Peak through the Trough (in +1 or 0)
ITARECDM	`Fred.OECDRecessionIndicators.ItalyFromPeakThroughTheTrough`	OECD based Recession Indicators for Italy from the Peak through the Trough (in +1 or 0)
JPNRECDM	`Fred.OECDRecessionIndicators.JapanFromPeakThroughTheTrough`	OECD based Recession Indicators for Japan from the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
KORRECDM	`Fred.OECDRecessionIndicators.KoreaFromPeakThroughTheTrough`	OECD based Recession Indicators for Korea from the Peak through the Trough (in +1 or 0)
LUXRECDM	`Fred.OECDRecessionIndicators.LuxembourgFromPeakThroughTheTrough`	OECD based Recession Indicators for Luxembourg from the Peak through the Trough (in +1 or 0)
MAJOR5ASIARECDM	`Fred.OECDRecessionIndicators.MajorFiveAsiaFromPeakThroughTheTrough`	OECD based Recession Indicators for Major 5 Asia from the Peak through the Trough (in +1 or 0)
MEXRECDM	`Fred.OECDRecessionIndicators.MexicoFromPeakThroughTheTrough`	OECD based Recession Indicators for Mexico from the Peak through the Trough (in +1 or 0)
MSCRECDM	`Fred.OECDRecessionIndicators.MajorSevenCountriesFromPeakThroughTheTrough`	OECD based Recession Indicators for Major Seven Countries from the Peak through the Trough (in +1 or 0)
NAFTARECDM	`Fred.OECDRecessionIndicators.NAFTAAreaFromPeakThroughTheTrough`	OECD based Recession Indicators for NAFTA Area from the Peak through the Trough (in +1 or 0)
NDLRECDM	`Fred.OECDRecessionIndicators.NetherlandsFromPeakThroughTheTrough`	OECD based Recession Indicators for Netherlands from the Peak through the Trough (in +1 or 0)
NORRECDM	`Fred.OECDRecessionIndicators.NorwayFromPeakThroughTheTrough`	OECD based Recession Indicators for Norway from the Peak through the Trough (in +1 or 0)
NZLRECDM	`Fred.OECDRecessionIndicators.NewZealandFromPeakThroughTheTrough`	OECD based Recession Indicators for New Zealand from the Peak through the Trough (in +1 or 0)
OECDEUROPERECDM	`Fred.OECDRecessionIndicators.OECDEuropeFromPeakThroughTheTrough`	OECD based Recession Indicators for OECD Europe from the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
OECDNMERECDM	`Fred.OECDRecessionIndicators.OECDAndNonmemberEconomiesFromPeakThroughTheTrough`	OECD based Recession Indicators for OECD and Non-member Economies from the Peak through the Trough (in +1 or 0)
OECDRECDM	`Fred.OECDRecessionIndicators.OECDTotalAreaFromPeakThroughTheTrough`	OECD based Recession Indicators for the OECD Total Area from the Peak through the Trough (in +1 or 0)
POLRECDM	`Fred.OECDRecessionIndicators.PolandFromPeakThroughTheTrough`	OECD based Recession Indicators for Poland from the Peak through the Trough (in +1 or 0)
PRTRECDM	`Fred.OECDRecessionIndicators.PortugalFromPeakThroughTheTrough`	OECD based Recession Indicators for Portugal from the Peak through the Trough (in +1 or 0)
RUSRECDM	`Fred.OECDRecessionIndicators.RussianFederationFromPeakThroughTheTrough`	OECD based Recession Indicators for Russian Federation from the Peak through the Trough (in +1 or 0)
SVKRECDM	`Fred.OECDRecessionIndicators.SlovakRepublicFromPeakThroughTheTrough`	OECD based Recession Indicators for the Slovak Republic from the Peak through the Trough (in +1 or 0)
SVNRECDM	`Fred.OECDRecessionIndicators.SloveniaFromPeakThroughTheTrough`	OECD based Recession Indicators for Slovenia from the Peak through the Trough (in +1 or 0)
SWERECDM	`Fred.OECDRecessionIndicators.SwedenFromPeakThroughTheTrough`	OECD based Recession Indicators for Sweden from the Peak through the Trough (in +1 or 0)
TURRECDM	`Fred.OECDRecessionIndicators.TurkeyFromPeakThroughTheTrough`	OECD based Recession Indicators for Turkey from the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
USARECDM	`Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough`	OECD based Recession Indicators for the United States from the Peak through the Trough (in +1 or 0)
ZAFRECDM	`Fred.OECDRecessionIndicators.SouthAfricaFromPeakThroughTheTrough`	OECD based Recession Indicators for South Africa from the Peak through the Trough (in +1 or 0)
4BIGEURORECD	`Fred.OECDRecessionIndicators.FourBigEuropeanCountriesFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Four Big European Countries from the Period following the Peak through the Trough (in +1 or 0)
AUSRECD	`Fred.OECDRecessionIndicators.AustraliaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Australia from the Period following the Peak through the Trough (in +1 or 0)
AUTRECD	`Fred.OECDRecessionIndicators.AustriaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Austria from the Period following the Peak through the Trough (in +1 or 0)
BELRECD	`Fred.OECDRecessionIndicators.BelgiumFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Belgium from the Period following the Peak through the Trough (in +1 or 0)
BRARECD	`Fred.OECDRecessionIndicators.BrazilFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Brazil from the Period following the Peak through the Trough (in +1 or 0)
CANRECD	`Fred.OECDRecessionIndicators.CanadaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Canada from the Period following the Peak through the Trough (in +1 or 0)
CHERECD	`Fred.OECDRecessionIndicators.SwitzerlandFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Switzerland from the Period following the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
CHLRECD	`Fred.OECDRecessionIndicators.ChileFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Chile from the Period following the Peak through the Trough (in +1 or 0)
CHNRECD	`Fred.OECDRecessionIndicators.ChinaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for China from the Period following the Peak through the Trough (in +1 or 0)
CZERECD	`Fred.OECDRecessionIndicators.CzechRepublicFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for the Czech Republic from the Period following the Peak through the Trough (in +1 or 0)
DEURECD	`Fred.OECDRecessionIndicators.GermanyFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Germany from the Period following the Peak through the Trough (in +1 or 0)
DNKRECD	`Fred.OECDRecessionIndicators.DenmarkFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Denmark from the Period following the Peak through the Trough (in +1 or 0)
ESPRECD	`Fred.OECDRecessionIndicators.SpainFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Spain from the Period following the Peak through the Trough (in +1 or 0)
ESTRECD	`Fred.OECDRecessionIndicators.EstoniaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Estonia from the Period following the Peak through the Trough (in +1 or 0)
EURORECD	`Fred.OECDRecessionIndicators.EuroAreaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Euro Area from the Period following the Peak through the Trough (in +1 or 0)
FINRECD	`Fred.OECDRecessionIndicators.FinlandFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Finland from the Period following the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
FRARECD	`Fred.OECDRecessionIndicators.FranceFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for France from the Period following the Peak through the Trough (in +1 or 0)
GBRRECD	`Fred.OECDRecessionIndicators.UnitedKingdomFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for the United Kingdom from the Period following the Peak through the Trough (in +1 or 0)
GRCRECD	`Fred.OECDRecessionIndicators.GreeceFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Greece from the Period following the Peak through the Trough (in +1 or 0)
HUNRECD	`Fred.OECDRecessionIndicators.HungaryFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Hungary from the Period following the Peak through the Trough (in +1 or 0)
IDNRECD	`Fred.OECDRecessionIndicators.IndonesiaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Indonesia from the Period following the Peak through the Trough (in +1 or 0)
INDRECD	`Fred.OECDRecessionIndicators.IndiaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for India from the Period following the Peak through the Trough (in +1 or 0)
IRLRECD	`Fred.OECDRecessionIndicators.IrelandFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Ireland from the Period following the Peak through the Trough (in +1 or 0)
ISRRECD	`Fred.OECDRecessionIndicators.IsraelFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Israel from the Period following the Peak through the Trough (in +1 or 0)
ITARECD	`Fred.OECDRecessionIndicators.ItalyFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Italy from the Period following the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
JPNRECD	`Fred.OECDRecessionIndicators.JapanFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Japan from the Period following the Peak through the Trough (in +1 or 0)
KORRECD	`Fred.OECDRecessionIndicators.KoreaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Korea from the Period following the Peak through the Trough (in +1 or 0)
LUXRECD	`Fred.OECDRecessionIndicators.LuxembourgFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Luxembourg from the Period following the Peak through the Trough (in +1 or 0)
MAJOR5ASIARECD	`Fred.OECDRecessionIndicators.MajorFiveAsiaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Major 5 Asia from the Period following the Peak through the Trough (in +1 or 0)
MEXRECD	`Fred.OECDRecessionIndicators.MexicoFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Mexico from the Period following the Peak through the Trough (in +1 or 0)
MSCRECD	`Fred.OECDRecessionIndicators.MajorSevenCountriesFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Major Seven Countries from the Period following the Peak through the Trough (in +1 or 0)
NAFTARECD	`Fred.OECDRecessionIndicators.NAFTAAreaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for NAFTA Area from the Period following the Peak through the Trough (in +1 or 0)
NDLRECD	`Fred.OECDRecessionIndicators.NetherlandsFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Netherlands from the Period following the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
NORRECD	`Fred.OECDRecessionIndicators.NorwayFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Norway from the Period following the Peak through the Trough (in +1 or 0)
NZLRECD	`Fred.OECDRecessionIndicators.NewZealandFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for New Zealand from the Period following the Peak through the Trough (in +1 or 0)
OECDEUROPERECD	`Fred.OECDRecessionIndicators.OECDEuropeFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for OECD Europe from the Period following the Peak through the Trough (in +1 or 0)
OECDNMERECD	`Fred.OECDRecessionIndicators.OECDandNonmemberEconomiesFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for OECD and Non-member Economies from the Period following the Peak through the Trough (in +1 or 0)
OECDRECD	`Fred.OECDRecessionIndicators.OECDTotalAreaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for the OECD Total Area from the Period following the Peak through the Trough (in +1 or 0)
POLRECD	`Fred.OECDRecessionIndicators.PolandFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Poland from the Period following the Peak through the Trough (in +1 or 0)
PRTRECD	`Fred.OECDRecessionIndicators.PortugalFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Portugal from the Period following the Peak through the Trough (in +1 or 0)
RUSRECD	`Fred.OECDRecessionIndicators.RussianFederationFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Russian Federation from the Period following the Peak through the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
SVKRECD	`Fred.OECDRecessionIndicators.SlovakRepublicFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for the Slovak Republic from the Period following the Peak through the Trough (in +1 or 0)
SVNRECD	`Fred.OECDRecessionIndicators.SloveniaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Slovenia from the Period following the Peak through the Trough (in +1 or 0)
SWERECD	`Fred.OECDRecessionIndicators.SwedenFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Sweden from the Period following the Peak through the Trough (in +1 or 0)
TURRECD	`Fred.OECDRecessionIndicators.TurkeyFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for Turkey from the Period following the Peak through the Trough (in +1 or 0)
USARECD	`Fred.OECDRecessionIndicators.UnitedStatesFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for the United States from the Period following the Peak through the Trough (in +1 or 0)
ZAFRECD	`Fred.OECDRecessionIndicators.SouthAfricaFromPeriodFollowingPeakThroughTheTrough`	OECD based Recession Indicators for South Africa from the Period following the Peak through the Trough (in +1 or 0)
4BIGEURORECDP	`Fred.OECDRecessionIndicators.FourBigEuropeanCountriesFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Four Big European Countries from the Peak through the Period preceding the Trough (in +1 or 0)
AUSRECDP	`Fred.OECDRecessionIndicators.AustraliaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Australia from the Peak through the Period preceding the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
AUTRECDP	`Fred.OECDRecessionIndicators.AustriaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Austria from the Peak through the Period preceding the Trough (in +1 or 0)
BELRECDP	`Fred.OECDRecessionIndicators.BelgiumFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Belgium from the Peak through the Period preceding the Trough (in +1 or 0)
BRARECDP	`Fred.OECDRecessionIndicators.BrazilFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Brazil from the Peak through the Period preceding the Trough (in +1 or 0)
CANRECDP	`Fred.OECDRecessionIndicators.CanadaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Canada from the Peak through the Period preceding the Trough (in +1 or 0)
CHERECDP	`Fred.OECDRecessionIndicators.SwitzerlandFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Switzerland from the Peak through the Period preceding the Trough (in +1 or 0)
CHLRECDP	`Fred.OECDRecessionIndicators.ChileFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Chile from the Peak through the Period preceding the Trough (in +1 or 0)
CHNRECDP	`Fred.OECDRecessionIndicators.ChinaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for China from the Peak through the Period preceding the Trough (in +1 or 0)
CZERECDP	`Fred.OECDRecessionIndicators.CzechRepublicFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for the Czech Republic from the Peak through the Period preceding the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
DEURECDP	`Fred.OECDRecessionIndicators.GermanyFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Germany from the Peak through the Period preceding the Trough (in +1 or 0)
DNKRECDP	`Fred.OECDRecessionIndicators.DenmarkFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Denmark from the Peak through the Period preceding the Trough (in +1 or 0)
ESPRECDP	`Fred.OECDRecessionIndicators.SpainFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Spain from the Peak through the Period preceding the Trough (in +1 or 0)
ESTRECDP	`Fred.OECDRecessionIndicators.EstoniaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Estonia from the Peak through the Period preceding the Trough (in +1 or 0)
EURORECDP	`Fred.OECDRecessionIndicators.EuroAreaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Euro Area from the Peak through the Period preceding the Trough (in +1 or 0)
FINRECDP	`Fred.OECDRecessionIndicators.FinlandFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Finland from the Peak through the Period preceding the Trough (in +1 or 0)
FRARECDP	`Fred.OECDRecessionIndicators.FranceFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for France from the Peak through the Period preceding the Trough (in +1 or 0)
GBRRECDP	`Fred.OECDRecessionIndicators.UnitedKingdomFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for the United Kingdom from the Peak through the Period preceding the Trough (in +1 or 0)
GRCRECDP	`Fred.OECDRecessionIndicators.GreeceFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Greece from the Peak through the Period preceding the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
HUNRECDP	`Fred.OECDRecessionIndicators.HungaryFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Hungary from the Peak through the Period preceding the Trough (in +1 or 0)
IDNRECDP	`Fred.OECDRecessionIndicators.IndonesiaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Indonesia from the Peak through the Period preceding the Trough (in +1 or 0)
INDRECDP	`Fred.OECDRecessionIndicators.IndiaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for India from the Peak through the Period preceding the Trough (in +1 or 0)
IRLRECDP	`Fred.OECDRecessionIndicators.IrelandFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Ireland from the Peak through the Period preceding the Trough (in +1 or 0)
ISRRECDP	`Fred.OECDRecessionIndicators.IsraelFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Israel from the Peak through the Period preceding the Trough (in +1 or 0)
ITARECDP	`Fred.OECDRecessionIndicators.ItalyFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Italy from the Peak through the Period preceding the Trough (in +1 or 0)
JPNRECDP	`Fred.OECDRecessionIndicators.JapanFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Japan from the Peak through the Period preceding the Trough (in +1 or 0)
KORRECDP	`Fred.OECDRecessionIndicators.KoreaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Korea from the Peak through the Period preceding the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
LUXRECDP	`Fred.OECDRecessionIndicators.LuxembourgFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Luxembourg from the Peak through the Period preceding the Trough (in +1 or 0)
MAJOR5ASIARECDP	`Fred.OECDRecessionIndicators.MajorFiveAsiaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Major 5 Asia from the Peak through the Period preceding the Trough (in +1 or 0)
MEXRECDP	`Fred.OECDRecessionIndicators.MexicoFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Mexico from the Peak through the Period preceding the Trough (in +1 or 0)
MSCRECDP	`Fred.OECDRecessionIndicators.MajorSevenCountriesFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Major Seven Countries from the Peak through the Period preceding the Trough (in +1 or 0)
NAFTARECDP	`Fred.OECDRecessionIndicators.NAFTAAreaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for NAFTA Area from the Peak through the Period preceding the Trough (in +1 or 0)
NDLRECDP	`Fred.OECDRecessionIndicators.NetherlandsFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Netherlands from the Peak through the Period preceding the Trough (in +1 or 0)
NORRECDP	`Fred.OECDRecessionIndicators.NorwayFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Norway from the Peak through the Period preceding the Trough (in +1 or 0)
NZLRECDP	`Fred.OECDRecessionIndicators.NewZealandFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for New Zealand from the Peak through the Period preceding the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
OECDEUROPERECDP	`Fred.OECDRecessionIndicators.OECDEuropeFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for OECD Europe from the Peak through the Period preceding the Trough (in +1 or 0)
OECDNMERECDP	`Fred.OECDRecessionIndicators.OECDandNonmemberEconomiesFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for OECD and Non-member Economies from the Peak through the Period preceding the Trough (in +1 or 0)
OECDRECDP	`Fred.OECDRecessionIndicators.OECDTotalAreaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for the OECD Total Area from the Peak through the Period preceding the Trough (in +1 or 0)
POLRECDP	`Fred.OECDRecessionIndicators.PolandFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Poland from the Peak through the Period preceding the Trough (in +1 or 0)
PRTRECDP	`Fred.OECDRecessionIndicators.PortugalFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Portugal from the Peak through the Period preceding the Trough (in +1 or 0)
RUSRECDP	`Fred.OECDRecessionIndicators.RussianFederationFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Russian Federation from the Peak through the Period preceding the Trough (in +1 or 0)
SVKRECDP	`Fred.OECDRecessionIndicators.SlovakRepublicFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for the Slovak Republic from the Peak through the Period preceding the Trough (in +1 or 0)
SVNRECDP	`Fred.OECDRecessionIndicators.SloveniaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Slovenia from the Peak through the Period preceding the Trough (in +1 or 0)

Symbol	Accessor Code	Summary
SWERECDP	`Fred.OECDRecessionIndicators.SwedenFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Sweden from the Peak through the Period preceding the Trough (in +1 or 0)
TURRECDP	`Fred.OECDRecessionIndicators.TurkeyFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for Turkey from the Peak through the Period preceding the Trough (in +1 or 0)
USARECDP	`Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for the United States from the Peak through the Period preceding the Trough (in +1 or 0)
ZAFRECDP	`Fred.OECDRecessionIndicators.SouthAfricaFromPeakThroughThePeriodPrecedingtheTrough`	OECD based Recession Indicators for South Africa from the Peak through the Period preceding the Trough (in +1 or 0)
TradeWeightedIndexes		
DTWEXM	`Fred.TradeWeightedIndexes.MajorCurrenciesGoods`	Trade Weighted U.S. Dollar Index: Major Currencies, Goods (in Index Mar 1973=100)
DTWEXO	`Fred.TradeWeightedIndexes.OtherImportantTradingPartnersGoods`	Trade Weighted U.S. Dollar Index: Other Important Trading Partners, Goods (in Index Jan 1997=100)
DTWEXB	`Fred.TradeWeightedIndexes.BroadGoods`	Trade Weighted U.S. Dollar Index: Broad, Goods (in Index Jan 1997=100)
DTWEXAFEGS	`Fred.TradeWeightedIndexes.AdvancedForeignEconomiesGoodsAndServices`	Trade Weighted U.S. Dollar Index: Advanced Foreign Economies, Goods and Services (in Index Jan 2006=100)
DTWEXBGS	`Fred.TradeWeightedIndexes.BroadGoodsAndServices`	Trade Weighted U.S. Dollar Index: Broad, Goods and Services (in Index Jan 2006=100)

Symbol	Accessor Code	Summary
DTWEXEMEGS	`Fred.TradeWeightedIndexes.EmergingMarketsEconomiesGoodsAndServices`	Trade Weighted U.S. Dollar Index: Emerging Markets Economies, Goods and Services (in Index Jan 2006=100)
Wilshire		
WILLSMLCAPVALPR	`Fred.Wilshire.USSmallCapValuePrice`	Wilshire US Small-Cap Value Price Index (in Index)
WILL2500PR	`Fred.Wilshire.Price2500`	Wilshire 2500 Price Index (in Index)
WILL4500PR	`Fred.Wilshire.Price4500`	Wilshire 4500 Price Index (in Index)
WILL2500PRVAL	`Fred.Wilshire.ValuePrice2500`	Wilshire 2500 Value Price Index (in Index)
WILL2500PRGR	`Fred.Wilshire.GrowthPrice2500`	Wilshire 2500 Growth Price Index (in Index)
WILLSMLCAPPR	`Fred.Wilshire.USSmallCapPrice`	Wilshire US Small-Cap Price Index (in Index)
WILL5000PR	`Fred.Wilshire.Price5000`	Wilshire 5000 Price Index (in Index)
WILLSMLCAPGRPR	`Fred.Wilshire.USSmallCapGrowthPrice`	Wilshire US Small-Cap Growth Price Index (in Index)
WILLMIDCAPVALPR	`Fred.Wilshire.USMidCapValuePrice`	Wilshire US Mid-Cap Value Price Index (in Index)
WILLRESIPR	`Fred.Wilshire.USRealEstateSecuritiesPrice`	Wilshire US Real Estate Securities Price Index (Wilshire US RESI) (in Index)
WILLLRGCAPPR	`Fred.Wilshire.USLargeCapPrice`	Wilshire US Large-Cap Price Index (in Index)
WILLMIDCAPPR	`Fred.Wilshire.USMidCapPrice`	Wilshire US Mid-Cap Price Index (in Index)
WILLMIDCAPGRPR	`Fred.Wilshire.USMidCapGrowthPrice`	Wilshire US Mid-Cap Growth Price Index (in Index)
WILLMICROCAPPR	`Fred.Wilshire.USMicroCapPrice`	Wilshire US Micro-Cap Price Index (in Index)

Symbol	Accessor Code	Summary
WILLREITPR	`Fred.Wilshire.USRealEstateInvestmentTrustPrice`	Wilshire US Real Estate Investment Trust Price Index (Wilshire US REIT) (in Index)
WILLLRGCAPVALPR	`Fred.Wilshire.USLargeCapValuePrice`	Wilshire US Large-Cap Value Price Index (in Index)
WILLLRGCAPGRPR	`Fred.Wilshire.USLargeCapGrowthPrice`	Wilshire US Large-Cap Growth Price Index (in Index)
WILL5000PRFC	`Fred.Wilshire.FullCapPrice5000`	Wilshire 5000 Full Cap Price Index (in Index)
WILLMIDCAPVAL	`Fred.Wilshire.USMidCapValue`	Wilshire US Mid-Cap Value Total Market Index (in Index)
WILLMIDCAPGR	`Fred.Wilshire.USMidCapGrowth`	Wilshire US Mid-Cap Growth Total Market Index (in Index)
WILLMIDCAP	`Fred.Wilshire.USMidCap`	Wilshire US Mid-Cap Total Market Index (in Index)
WILLRESIND	`Fred.Wilshire.USRealEstateSecurities`	Wilshire US Real Estate Securities Total Market Index (Wilshire US RESI) (in Index)
WILL4500IND	`Fred.Wilshire.Index4500`	Wilshire 4500 Total Market Index (in Index)
WILL5000IND	`Fred.Wilshire.Index5000`	Wilshire 5000 Total Market Index (in Index)
WILLLRGCAPGR	`Fred.Wilshire.USLargeCapGrowth`	Wilshire US Large-Cap Growth Total Market Index (in Index)
WILLMICROCAP	`Fred.Wilshire.USMicroCap`	Wilshire US Micro-Cap Total Market Index (in Index)
WILL2500INDVAL	`Fred.Wilshire.Value2500`	Wilshire 2500 Value Total Market Index (in Index)
WILLSMLCAPGR	`Fred.Wilshire.USSmallCapGrowth`	Wilshire US Small-Cap Growth Total Market Index (in Index)
WILLSMLCAPVAL	`Fred.Wilshire.USSmallCapValue`	Wilshire US Small-Cap Value Total Market Index (in Index)
WILLLRGCAPVAL	`Fred.Wilshire.USLargeCapValue`	Wilshire US Large-Cap Value Total Market Index (in Index)

Symbol	Accessor Code	Summary
WILLREITIND	`Fred.Wilshire.USRealEstateInvestmentTrust`	Wilshire US Real Estate Investment Trust Total Market Index (Wilshire US REIT) (in Index)
WILL2500IND	`Fred.Wilshire.Index2500`	Wilshire 2500 Total Market Index (in Index)
WILLSMLCAP	`Fred.Wilshire.USSmallCap`	Wilshire US Small-Cap Total Market Index (in Index)
WILLLRGCAP	`Fred.Wilshire.USLargeCap`	Wilshire US Large-Cap Total Market Index (in Index)
WILL2500INDGR	`Fred.Wilshire.Growth2500`	Wilshire 2500 Growth Total Market Index (in Index)
WILL5000INDFC	`Fred.Wilshire.TotalMarketFullCap5000`	Wilshire 5000 Total Market Full Cap Index (in Index)

Data Point Attributes

The FRED dataset provides **Fred** objects, which have the following attributes:

Requesting Data

To add FRED data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class FredAlternativeDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2003, 1, 1)
        self.SetEndDate(2019, 10, 11)
        self.SetCash(100000)

        self.dataset_symbol = self.AddData(Fred,
Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough, Resolution.Daily).Symbol
```

Accessing Data

To get the current FRED data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} value at {slice.Time}: {data_point.Value}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(Fred).items():
        self.Log(f"{dataset_symbol} value at {slice.Time}: {data_point.Value}")
```

Historical Data

To get historical FRED data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[Fred](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

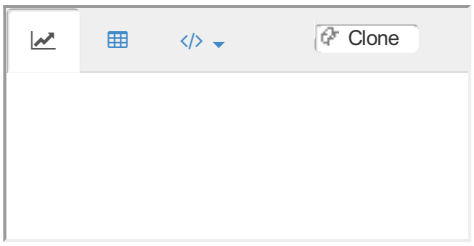
To remove your subscription to FRED data, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The FRED dataset enables you to accurately design strategies utilizing macroeconomic indicators. Examples include the following strategies:

- Trading on macroeconomic factors
- Macroeconomic risk modeling


Clone

Datasets

Kavout

Kavout was created by ex-Googlers and the founding team used to work at Google, Microsoft, Baidu, and financial firms with a proven track record of building many mission-critical machine learning systems where billions of data points were processed in real-time to predict the best outcome for core search ranking, ads monetization, recommendations, and trading platforms.

Their mission is to build machine investing solutions to find alpha with adaptive learning algorithms and to create an edge by assimilating vast quantities of complex data through the latest AI and Machine Learning methods to generate signals to uncover hidden, dynamic, and nonlinear patterns in the financial markets.

Composite Factor Bundle

Kavout

Composite Factor Bundle

Introduction

The Composite Factor Bundle dataset by Kavout provides ensemble scores for popular market factors. Kavout signals are machine-learning enhanced scores that capture the returns of systematic factors such as quality, value, momentum, growth, and low volatility. There are many different anomalies discovered by researchers and practitioners across these factor categories and there is no good common definition of each style across the literature. Kavout creates an ensemble score for each style that gauges the different factors considered in the literature and industry practice.

In this data set, you will find Kavout's proprietary signals for quality, value, momentum, growth, and low volatility, which have been adopted by some of the multi-billion dollar quant funds in New York and London. Each signal is generated by an ensemble model consisting of inputs from hundreds of anomalies. The data is generated on a daily basis and covers all the stocks traded in US major markets such as NYSE and Nasdaq since 2003. You could leverage this abundant set of signals to construct and backtest your strategies.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Composite Factor Bundle dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Kavout was created by ex-Googlers and the founding team used to work at Google, Microsoft, Baidu, and financial firms with a proven track record of building many mission-critical machine learning systems where billions of data points were processed in real-time to predict the best outcome for core search ranking, ads monetization, recommendations, and trading platforms.

Their mission is to build machine investing solutions to find alpha with adaptive learning algorithms and to create an edge by assimilating vast quantities of complex data through the latest AI and Machine Learning methods to generate signals to uncover hidden, dynamic, and nonlinear patterns in the financial markets.

Getting Started

The following snippet demonstrates how to request data from the Composite Factor Bundle dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(KavoutCompositeFactorBundle, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2003
Asset Coverage	8,000 US Equities
Data Density	Regular
Resolution	Daily
Timezone	UTC

Data Point Attributes

Template content example

The Composite Factor Bundle dataset provides **KavoutCompositeFactorBundle** objects, which have the following attributes:

Requesting Data

To add Composite Factor Bundle data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class KavoutCompositeFactorBundleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(KavoutCompositeFactorBundle, self.symbol).Symbol
```

Accessing Data

To get the current Composite Factor Bundle data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} momentum at {slice.Time}: {data_point.Momentum}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(KavoutCompositeFactorBundle).items():
        self.Log(f"{dataset_symbol} momentum at {slice.Time}: {data_point.Momentum}")
```

Historical Data

To get historical Composite Factor Bundle data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[KavoutCompositeFactorBundle](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

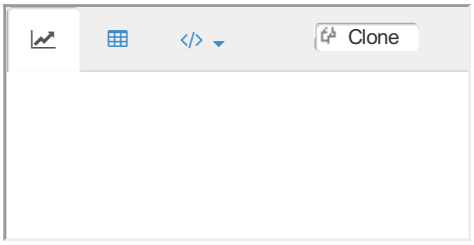
```
PY
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Composite Factor Bundle data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Composite Factor Bundle dataset enables you to access the performance of 5 different factors in order to engineer strategies. Examples include the following strategies:

- Performing return-risk optimization based on performance and volatility scoring.
- Weighing stocks based on regression analysis in factor-vector space.



Datasets

Nasdaq

" Nasdaq " was initially an acronym for the National Association of Securities Dealers Automated Quotations. It was founded in 1971 by the National Association of Securities Dealers (NASD), now known as the Financial Industry Regulatory Authority (FINRA), with the goal to provide financial services and operate stock exchanges.

On Dec. 4th, 2018, Nasdaq announced it had acquired Quandl, Inc. , a leading provider of alternative and core financial data. Quandl was founded by Tammer Kamel in 2012, with goal of making it easy for anyone to find and use high-quality data effectively in their professional decision-making. In 2021, Quandl was replaced by Nasdaq Data Link .

Data Link

Nasdaq

Data Link

Introduction

The Data Link dataset by Nasdaq, previously known as Quandl, is a collection of alternative data sets. It has indexed millions of time-series datasets from over 400 sources, which start in different years. This dataset is delivered on several frequencies, but the free data sets have often a daily frequency.

For more information about the Data Link dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

" Nasdaq " was initially an acronym for the National Association of Securities Dealers Automated Quotations. It was founded in 1971 by the National Association of Securities Dealers (NASD), now known as the Financial Industry Regulatory Authority (FINRA), with the goal to provide financial services and operate stock exchanges.

On Dec. 4th, 2018, Nasdaq announced it had acquired Quandl, Inc. , a leading provider of alternative and core financial data. Quandl was founded by Tammer Kamel in 2012, with goal of making it easy for anyone to find and use high-quality data effectively in their professional decision-making. In 2021, Quandl was replaced by Nasdaq Data Link .

Getting Started

The following snippet demonstrates how to request data from the Data Link dataset:

```
PY
from QuantConnect.DataSource import *

# For premium datasets, provide your API Key
# NasdaqDataLink.SetAuthCode(self.GetParameter("nasdaq-data-link-api-key"))

self.pe_ratio_symbol = self.AddData(NasdaqDataLink, "MULTPL/SP500_PE_RATIO_MONTH",
Resolution.Daily).Symbol
# This dataset has one data column ("Value")

self.position_change_symbol = self.AddData(NasdaqCustomColumns, "CFTC/066393_FO_L_CHG",
Resolution.Daily).Symbol
# This dataset has multiple data columns. Create a subclass to set the default value column.

class NasdaqCustomColumns(NasdaqDataLink):
    def __init__(self) -> None:
        # Select the column "open interest - change".
        self.ValueColumnName = "open interest - change"
```

Our Nasdaq Data Link integration supports any dataset from Nasdaq that has a URL starting with **data.nasdaq.com/data/** . To import a dataset from Data Link, you need to get the dataset code. Follow these steps to get the dataset code:

1. Open the Data Link catalog .
2. *(Optional)* Use the filters on the left side bar to narrow down the catalog results.
3. Click one of the data products.

4. On the data product page, click one of the table names.
5. In the top-right corner of the dataset page, copy the Nasdaq Data Link Code.

Data Summary

The data summary depends on the specific Data Link dataset you use. Follow these steps to view the data summary of a dataset:

1. Open the Data Link catalog .
2. Click one of the data products.
3. On the data product page, click one of the table names.
4. View the data frequency and description in the left sidebar.
5. Above the chart, click **Table** to view the start date, end date, and data point attributes.

Backward Compatibility

QuantConnect/LEAN has supported Quandl since 2015. On January, 12, 2022, we moved the implementation from the LEAN GitHub repository to the Lean.DataSource.NasdaqDataLink GitHub repository . Through this transition, we maintained backward compatibility. All of the preceding code snippets work if you replace **NasdaqDataLink** with **Quandl** .

Requesting Data

To add Data Link data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm. If there is more than one value column in the Data Link dataset, to set the **Value** property of the data objects, create a sublcass of the **NasdaqDataLink** class and set its **ValueColumnName** property to the column name.

```
PY
class NasdaqDataLinkDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2022, 1, 1)
        self.SetEndDate(2022, 7, 1)
        self.SetCash(100000)

        # For premium datasets, provide your API Key
        # NasdaqDataLink.SetAuthCode(self.GetParameter("nasdaq-data-link-api-key"))

        self.pe_ratio_symbol = self.AddData(NasdaqDataLink, "MULTPL/SP500_PE_RATIO_MONTH",
Resolution.Daily).Symbol
        # This dataset has one data column ("Value")
        # Source : https://data.nasdaq.com/data/MULTPL/SP500_PE_RATIO_MONTH-sp-500-pe-ratio-by-month

        self.position_change_symbol = self.AddData(NasdaqCustomColumns, "CFTC/066393_FO_L_CHG",
Resolution.Daily).Symbol
        # This dataset has multiple data columns
        # Source: https://data.nasdaq.com/data/CFTC/066393_FO_L_CHG-commitment-of-traders-propane-argus-
far-east-mini-ifed-futures-and-options-change-in-positions-legacy-format-066393

class NasdaqCustomColumns(NasdaqDataLink):
    def __init__(self) -> None:
        # Select the column "open interest - change".
        self.ValueColumnName = "open interest - change"
```

Accessing Data

To get the current Data Link data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid

issues, check if the **Slice** contains the data you want before you index it.

To get the default value of the dataset, use the **Value** property. To get the value of a specific column in the dataset, call the **GetStorageDictionary** method and index the result with the column name.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.pe_ratio_symbol):
        value = slice[self.pe_ratio_symbol].Value
        self.Log(f"{self.pe_ratio_symbol} PE ratio at {slice.Time}: {value}")

    if slice.ContainsKey(self.position_change_symbol):
        data_point = slice[self.position_change_symbol]
        open_interest_change = data_point.Value
        storage_dictionary = data_point.GetStorageDictionary()
        longs_change = storage_dictionary["noncommercial longs - change"]
        self.Log(f"{self.position_change_symbol} open interest change at {slice.Time}:
{open_interest_change}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(NasdaqDataLink).items():
        self.Log(f"{dataset_symbol} PE ratio at {slice.Time}: {data_point.Value}")

    for dataset_symbol, data_point in slice.Get(NasdaqCustomColumns).items():
        self.Log(f"{dataset_symbol} value at {slice.Time}: {data_point.Value}")
```

Historical Data

The process to get historical Data Link data depends on your environment.

In Algorithms

To get historical Data Link data in an algorithm, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrames
pe_ratio_history_df = self.History(self.pe_ratio_symbol, 100, Resolution.Daily)
position_change_history_df = self.History(self.position_change_symbol, 100, Resolution.Daily)

# Dataset objects
pe_ratio_history_df = self.History[NasdaqDataLink](self.pe_ratio_symbol, 100, Resolution.Daily)
position_change_history_df = self.History[NasdaqCustomColumns](self.position_change_symbol, 100,
Resolution.Daily)
```

For more information about historical data, see History Requests .

In Research Notebooks

To get historical Data Link data in the Research Environment, call the **Download** method with the data URL.

```
PY
from io import StringIO
data = qb.Download("data.nasdaq.com/api/v3/datasets/MULTPL/SP500_PE_RATIO_MONTH.csv?")
df = pd.read_csv(StringIO(data), index_col=0)
```

To receive a notification when you can use the **History** method in the Research Environment to get data from Data Link, subscribe to Lean.DataSource.NasdaqDataLink GitHub Issue #10 .

Remove Subscriptions

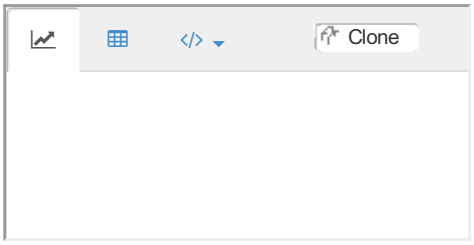
To remove a subscription, call the **RemoveSecurity** method.

```
PY
self.RemoveSecurity(self.pe_ratio_symbol)
self.RemoveSecurity(self.position_change_symbol)
```

Example Applications

The Nasdaq Data Link sources allow you to explore different kinds of data in their database to develop trading strategies. Examples include the following strategies:

- Using alternative data to regress market regime/asset price.
- Backtesting exotic derivatives or private Equity investments.



Datasets

Quiver Quantitative

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

CNBC Trading

Corporate Lobbying

Insider Trading

Twitter Followers

US Congress Trading

US Government Contracts

WallStreetBets

Wikipedia Page Views

Quiver Quantitative

CNBC Trading

Introduction

The CNBC Trading dataset by Quiver Quantitative tracks the recommendations made by media personalities on CNBC and their historical performance. The data covers over 1,500 US Equities, starts in December 2020, and is delivered on a daily frequency. This dataset covers recommendations made on Mad Money, Halftime Report, and Fast Money.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the CNBC Trading dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the CNBC Trading dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverCNBCs, self.symbol).Symbol

self.AddUniverse(QuiverCNBCsUniverse, "QuiverCNBCsUniverse", Resolution.Daily,
self.UniverseSelectionFilter)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	December 25, 2020
Asset Coverage	1,515 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Quiver Quantitative CNBC Trading dataset provides **QuiverCNBCs** , **QuiverCNBC** , and **QuiverCNBCsUniverse** objects.

QuiverCNBCs

QuiverCNBCs objects have the following attributes:

QuiverCNBC

QuiverCNBC objects have the following attributes:

QuiverCNBCsUniverse

QuiverCNBCsUniverse objects have the following attributes:

Requesting Data

To add CNBC Trading data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class QuiverCNBCDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverCNBCs, self.symbol).Symbol
```

Accessing Data

To get the current CNBC Trading data, index the current **Slice** with the dataset **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_points = slice[self.dataset_symbol]
        for data_point in data_points:
            self.Log(f"{self.dataset_symbol} direction at {slice.Time}: {data_point.Direction}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_points in slice.Get(QuiverCNBCs).items():
        for data_point in data_points:
            self.Log(f"{dataset_symbol} direction at {slice.Time}: {data_point.Direction}")
```

Historical Data

To get historical CNBC Trading data, call the **History** method with the dataset **Symbol** . If there is no data in the

period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverCNBCs](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on CNBC Trading data, call the **AddUniverse** method with the **QuiverCNBCsUniverse** class and a selection function.

```
self.AddUniverse(QuiverCNBCsUniverse, "QuiverCNBCsUniverse", Resolution.Daily, self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverCNBCsUniverse]) -> List[Symbol]:
    cnbc_data_by_symbol = {}

    for datum in alt_coarse:
        symbol = datum.Symbol

        if symbol not in cnbc_data_by_symbol:
            cnbc_data_by_symbol[symbol] = []
        cnbc_data_by_symbol[symbol].append(datum)

    # define our selection criteria
    return [symbol for symbol, d in cnbc_data_by_symbol.items()
            if len([x for x in d if x.Direction == OrderDirection.Buy]) >= 3]
```

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to CNBC Trading data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Quiver Quantitative CNBC Trading dataset enables you to create strategies using the latest recommendations made by media personalities on CNBC. Examples include the following strategies:

- Taking short positions in securities that were mentioned by Jim Cramer (CNBC commentator) in the last week
- Trading securities that were most/least discussed across CNBC programs over the last year

Quiver Quantitative

Corporate Lobbying

Introduction

The Corporate Lobbying dataset by Quiver Quantitative tracks the lobbying activity of US Equities. The Lobbying Disclosure Act of 1995 requires lobbyists in the United States to disclose information about their activities, such as their clients, which issues they are lobbying on, and how much they are being paid. Quiver Quantiative scrapes this data and maps it to stock tickers to track which companies are spending money for legislative influence.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Corporate Lobbying dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the Corporate Lobbying dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverLobbyings, self.symbol).Symbol

self.AddUniverse(QuiverLobbyingUniverse, "QuiverLobbyingUniverse", Resolution.Daily,
self.UniverseSelectionFilter)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 4, 1999
Asset Coverage	1,418 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Quiver Quantitative Corporate Lobbying dataset provides **QuiverLobbyings** , **QuiverLobbying** , and **QuiverLobbyingUniverse** objects.

QuiverLobbyings

QuiverLobbyings objects have the following attributes:

QuiverLobbying

QuiverLobbying objects have the following attributes:

QuiverLobbyingUniverse

QuiverLobbyingUniverse objects have the following attributes:

Requesting Data

To add Corporate Lobbying data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class QuiverLobbyingDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverLobbyings, self.symbol).Symbol
```

PY

Accessing Data

To get the current Corporate Lobbying data, index the current Slice with the dataset **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_points = slice[self.dataset_symbol]
        for data_point in data_points:
            self.Log(f"{self.dataset_symbol} amount at {slice.Time}: {data_point.Amount}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_points in slice.Get(QuiverLobbyings).items():
        for data_point in data_points:
            self.Log(f"{dataset_symbol} amount at {slice.Time}: {data_point.Amount}")
```

Historical Data

To get historical Corporate Lobbying data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverLobbyings](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Corporate Lobbying data, call the **AddUniverse** method with the **QuiverLobbyingUniverse** class and a selection function.

```
self.AddUniverse(QuiverLobbyingUniverse, "QuiverLobbyingUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverLobbyingUniverse]) -> List[Symbol]:
    lobby_data_by_symbol = {}

    for datum in alt_coarse:
        symbol = datum.Symbol

        if symbol not in lobby_data_by_symbol:
            lobby_data_by_symbol[symbol] = []
        lobby_data_by_symbol[symbol].append(datum)

    return [symbol for symbol, d in lobby_data_by_symbol.items()
            if sum([x.Amount for x in d]) >= 100000]
```

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Corporate Lobbying data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Corporate Lobbying dataset enables you to create strategies using the latest information on lobbying activity. Examples include the following strategies:

- Trading securities that have spent the most on lobbying over the last quarter
- Trading securities that have had the biggest change in lobbying spend for privacy legislation over the last year

Quiver Quantitative

Insider Trading

Introduction

Corporate insiders are required to disclose purchases or sales of their own stock within two business days of when they occur. Using these disclosures, we collect data on insider trading activity, which can give hints on whether executives are bullish or bearish on their own companies. Here is a blog that we did on this dataset: https://www.quiverquant.com/blog/081121

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Insider Trading dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the Insider Trading dataset:

```
PY
from QuantConnect.DataSource import *

aapl = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.quiver_insider_trading_symbol = self.AddData(QuiverInsiderTradings, aapl).Symbol
self.AddUniverse(QuiverInsiderTradingUniverse, "QuiverInsiderTradingUniverse", Resolution.Daily,
self.UniverseSelection)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	25 April 2014
Asset Coverage	4994 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Insider Trading dataset provides **QuiverInsiderTrading** objects encapsulated in a **QuiverInsiderTradings** object, and **QuiverInsiderTradingUniverse** object.

QuiverInsiderTradings

QuiverInsiderTradings objects have the following attributes:

QuiverInsiderTrading

QuiverInsiderTrading objects have the following attributes:

QuiverInsiderTradingUniverse

QuiverInsiderTradingUniverse objects have the following attributes:

Requesting Data

To add Insider Trading data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class QuiverInsiderTradingDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverInsiderTradings, self.symbol).Symbol
```

Accessing Data

To get the current Insider Trading data, index the current **Slice** with the dataset **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_points = slice[self.dataset_symbol]
        for data_point in data_points:
            self.Log(f"{self.dataset_symbol} shares at {slice.Time}: {data_point.Shares}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_points in slice.Get(QuiverInsiderTradings).items():
        for data_point in data_points:
            self.Log(f"{dataset_symbol} shares at {slice.Time}: {data_point.Shares}")
```

Historical Data

To get historical Insider Trading data, call the **History** method with the dataset **Symbol** . If there is no data in the

period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
self.History[QuiverInsiderTradings](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Insider Trading data, call the **AddUniverse** method with the **QuiverInsiderTradingUniverse** class and a selection function.

```
self.AddUniverse(QuiverInsiderTradingUniverse, "QuiverInsiderTradingUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverInsiderTradingUniverse]) -> List[Symbol]:
    insider_trading_data_by_symbol = {}

    for datum in alt_coarse:
        symbol = datum.Symbol

        if symbol not in insider_trading_data_by_symbol:
            insider_trading_data_by_symbol[symbol] = []
        insider_trading_data_by_symbol[symbol].append(datum)

    return [symbol for symbol, d in insider_trading_data_by_symbol.items()
            if len([x for x in d if x.Direction == OrderDirection.Buy]) >= 3]
```

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Insider Trading data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Quiver Quantitative Insider Trading dataset enables researchers to create strategies using the latest information on insider trading activity. Examples include:

- Taking a short position in securities that have had the most insider selling over the last 5 days
- Buying any security that has had over $100,000 worth of shares purchased by insiders in the last month

Quiver Quantitative

Twitter Followers

Introduction

The Twitter Followers dataset by Quiver Quantitative tracks the number of followers on the official Twitter pages of US-listed companies. The data covers 2,000 equities, starts in May 2020, and is delivered on a daily frequency. This dataset is created by scraping the number of Twitter followers from the official Twitter page of the security.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Twitter Followers dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the Twitter Followers dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverQuantTwitterFollowers, self.symbol).Symbol

self.AddUniverse(QuiverQuantTwitterFollowersUniverse, "QuiverQuantTwitterFollowersUniverse",
Resolution.Daily, self.UniverseSelectionMethod)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	May 2020
Asset Coverage	2,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	Chicago

Data Point Attributes

The Twitter Followers dataset provides **QuiverQuantTwitterFollowers** and **QuiverQuantTwitterFollowersUniverse** objects.

QuiverQuantTwitterFollowers Attributes

QuiverQuantTwitterFollowers objects have the following attributes:

QuiverQuantTwitterFollowersUniverse Attributes

QuiverQuantTwitterFollowersUniverse objects have the following attributes:

Requesting Data

To add Twitter Followers data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class QuiverQuantTwitterFollowersDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverQuantTwitterFollowers, self.symbol).Symbol
```

Accessing Data

To get the current Twitter Followers data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} followers at {slice.Time}: {data_point.Followers}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(QuiverQuantTwitterFollowers).items():
        self.Log(f"{dataset_symbol} followers at {slice.Time}: {data_point.Followers}")
```

Historical Data

To get historical Twitter Followers data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverQuantTwitterFollowers](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Twitter Followers data, call the **AddUniverse** method with the **QuiverQuantTwitterFollowersUniverse** class and a selection function.

```
def Initialize(self) -> None:
    self.AddUniverse(QuiverQuantTwitterFollowersUniverse, "QuiverQuantTwitterFollowersUniverse",
Resolution.Daily, self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverQuantTwitterFollowersUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.Followers > 200000 \
                and d.WeekPercentChange > 0]
```

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

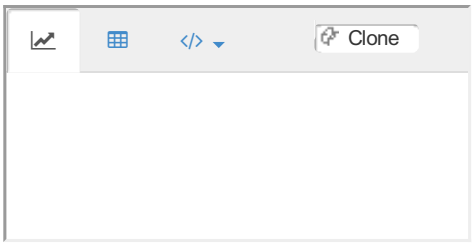
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Twitter Followers data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Twitter Followers dataset enables you to create strategies using the number of Twitter followers for companies. Examples include the following strategies:

- Trading securities that are on a upward/downward trend for number of followers
- Trading the security that has the highest/lowest increase in followers on a given day
- Trading securities with big changes in their follower count, prices, and volume


Clone

Quiver Quantitative

US Congress Trading

Introduction

The US Congress Trading dataset by Quiver Quantitative tracks US Equity trades made by members of Congress in the Senate and the House of Representatives. The data covers 1,800 US Equities, starts in January 2016, and is delivered on a daily frequency. This dataset is created by scraping SEC reports.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Congress Trading dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the US Congress Trading dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverCongress, self.symbol).Symbol

self.AddUniverse(QuiverQuantCongressUniverse, "QuiverQuantCongresssUniverse", Resolution.Daily,
self.UniverseSelection)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2016
Asset Coverage	1,800 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The US Congress Trading dataset provides **QuiverCongress** and **QuiverQuantCongressUniverse** objects.

QuiverCongress Attributes

QuiverCongress object has the following attributes:

QuiverQuantCongressUniverse Attributes

QuiverQuantCongressUniverse object has the following attributes:

Requesting Data

To add US Congress Trading data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class QuiverCongressDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverCongress, self.symbol).Symbol
```

Accessing Data

To get the current US Congress Trading data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} transaction amount at {slice.Time}: {data_point.Amount}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(QuiverCongress).items():
        self.Log(f"{dataset_symbol} transaction amount at {slice.Time}: {data_point.Amount}")
```

Historical Data

To get historical US Congress Trading data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverCongress](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on US Congress Trading data, call the **AddUniverse** method with the **QuiverQuantCongressUniverse** class and a selection function.

```
def Initialize(self) -> None:
    self.AddUniverse(QuiverQuantCongressUniverse, "QuiverQuantCongresssUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverQuantCongresssUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
        if d.Amount > 200000 and d.Transaction == OrderDirection.Buy]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

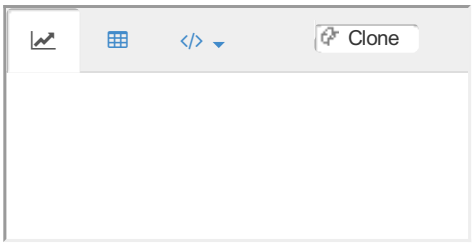
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to US Congress Trading data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The US Congress Trading dataset enables you to take immediate action on trades made by informed Members of Congress. Examples include the following strategies:

- Following the trades of specific representatives on the premise that the representatives are more informed
- Assigning a long/short-bias to securities on a daily frequency based on how Members of Congress are trading them


Clone

Quiver Quantitative

US Government Contracts

Introduction

The US Government Contracts dataset by Quiver Quantitative tracks the transactions of government contracts with publicly traded companies. The data covers over 700 US Equities, starts in October 2019, and is delivered on a daily frequency. Quiver Quantitative creates this dataset by using the API for USASpending.gov , which has the official open data source of federal spending information. The rows in this dataset only show new contracts, not payments or modifications to existing contracts. The dollar amounts are based on the total dollars obligated from each contract.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Government Contracts dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the US Government Contracts dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverGovernmentContracts, self.symbol).Symbol

self.AddUniverse(QuiverGovernmentContractUniverse, "QuiverGovernmentContractUniverse", Resolution.Daily,
self.UniverseSelectionFilter)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 15, 2019
Asset Coverage	748 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The US Government Contracts dataset provides **QuiverGovernmentContracts** , **QuiverGovernmentContract** , and **QuiverGovernmentContractUniverse** objects.

QuiverGovernmentContracts

QuiverGovernmentContracts objects have the following attributes:

QuiverGovernmentContract

QuiverGovernmentContract objects have the following attributes:

QuiverGovernmentContractUniverse

QuiverGovernmentContractUniverse objects have the following attributes:

Requesting Data

To add US Government Contracts data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class QuiverGovernmentContractsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverGovernmentContracts, self.symbol).Symbol
```

PY

Accessing Data

To get the current US Government Contract data, index the current **Slice** with the dataset **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_points = slice[self.dataset_symbol]
        for data_point in data_points:
            self.Log(f"{self.dataset_symbol} amount at {slice.Time}: {data_point.Amount}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY

def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_points in slice.Get(QuiverGovernmentContracts).items():
        for data_point in data_points:
            self.Log(f"{dataset_symbol} amount at {slice.Time}: {data_point.Amount}")
```

Historical Data

To get historical US Government Contracts data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY

# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverGovernmentContracts](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on US Government Contract data, call the **AddUniverse** method with the **QuiverGovernmentContractUniverse** class and a selection function.

```
PY

self.AddUniverse(QuiverGovernmentContractUniverse, "QuiverGovernmentContractUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coase: List[QuiverGovernmentContractUniverse]) -> List[Symbol]:
    gov_contract_data_by_symbol = {}

    for datum in alt_coarse:
        symbol = datum.Symbol

        if symbol not in gov_contract_data_by_symbol:
            gov_contract_data_by_symbol[symbol] = []
        gov_contract_data_by_symbol[symbol].append(datum)

    return [symbol for symbol, d in gov_contract_data_by_symbol.items()
            if len(d) >= 3 and sum([x.Amount for x in d]) > 50000]
```

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

```
PY

self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to US Government Contracts data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Quiver Quantitative US Government Contracts dataset enables you to create strategies using the latest information on government contracts activity. Examples include the following strategies:

- Buying securities that have received the most new government contracts awards over the last month
- Trading securities that have had the biggest change in government contracts awards over the last year

Quiver Quantitative

WallStreetBets

Introduction

The WallStreetBets dataset by Quiver Quantitative tracks daily mentions of different equities on Reddit's popular WallStreetBets forum. The data covers 6,000 Equities, starts in August 2018, and is delivered on a daily frequency. The dataset is created by scraping the daily discussion threads on r/WallStreetBets and parsing the comments for ticker mentions.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the WallStreetBets dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the WallStreetBets dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverWallStreetBets, self.symbol).Symbol

self.AddUniverse(QuiverWallStreetBetsUniverse, "QuiverWallStreetBetsUniverse", Resolution.Daily,
self.UniverseSelection)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2018
Asset Coverage	6,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The WallStreetBets dataset provides **QuiverWallStreetBets** and **QuiverWallStreetBetsUniverse** objects.

QuiverWallStreetBets Attributes

QuiverWallStreetBets objects have the following attributes:

QuiverWallStreetBetsUniverse Attributes

QuiverWallStreetBetsUniverse objects have the following attributes:

Requesting Data

To add WallStreetBets data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY

class QuiverWallStreetBetsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverWallStreetBets, self.symbol).Symbol
```

Accessing Data

To get the current WallStreetBets data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} mentions at {slice.Time}: {data_point.Mentions}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY

def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(QuiverWallStreetBets).items():
        self.Log(f"{dataset_symbol} mentions at {slice.Time}: {data_point.Mentions}")
```

Historical Data

To get historical WallStreetBets data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverWallStreetBets](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on WallStreetBets data, call the **AddUniverse** method with the **QuiverWallStreetBetsUniverse** class and a selection function.

```
def Initialize(self) -> None:
    self.AddUniverse(QuiverWallStreetBetsUniverse, "QuiverWallStreetBetsUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverWallStreetBetsUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.Mentions > 100 \
                and d.Rank < 100]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

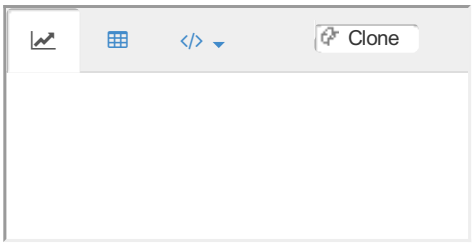
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to WallStreetBets data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The WallStreetBets dataset enables you to create strategies using the latest activity on the WallStreetBets daily discussion thread. Examples include the following strategies:

- Trading any security that is being mentioned
- Trading securities that are receiving more/less mentions than they were previously
- Trading the security that is being mentioned the most/least for the day


Clone

Quiver Quantitative

Wikipedia Page Views

Introduction

The Wikipedia Page Views dataset by Quiver Quantitative tracks Wikipedia page views for US Equities. The data covers 1,300 US Equities, starts in October 2016, and is delivered on a daily frequency. This dataset is created by scraping the Wikipedia pages of companies.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Wikipedia Page Views dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the Wikipedia Page Views dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
self.dataset_symbol = self.AddData(QuiverWikipedia, self.symbol).Symbol

self.AddUniverse(QuiverWikipediaUniverse, "QuiverWikipediaUniverse", Resolution.Daily,
self.UniverseSelectionMethod)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 2016
Asset Coverage	1,300 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The Wikipedia Page Views dataset provides **QuiverWikipedia** and **QuiverWikipediaUniverse** objects.

QuiverWikipedia Attributes

QuiverWikipedia objects have the following attributes:

QuiverWikipediaUniverse Attributes

QuiverWikipediaUniverse objects have the following attributes:

Requesting Data

To add Wikipedia Page Views data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class QuiverWikipediaPageViewsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Daily).Symbol
        self.dataset_symbol = self.AddData(QuiverWikipedia, self.symbol).Symbol
```

Accessing Data

To get the current Wikipedia Page Views data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} weekly page views percentage change at {slice.Time}:
{data_point.WeekPercentChange}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(QuiverWikipedia).items():
        self.Log(f"{dataset_symbol} weekly page views percentage change at {slice.Time}:
{data_point.WeekPercentChange}")
```

Historical Data

To get historical Wikipedia Page Views data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[QuiverWikipedia](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Wikipedia Page Views data, call the **AddUniverse** method with the **QuiverWikipediaUniverse** class and a selection function.

```
def Initialize(self) -> None:
    self.AddUniverse(QuiverWikipediaUniverse, "QuiverWikipediaUniverse", Resolution.Daily,
self.UniverseSelection)

def UniverseSelection(self, alt_coarse: List[QuiverWikipediaUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.PageViews > 100 \
                and d.WeekPercentChange < 0.2]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

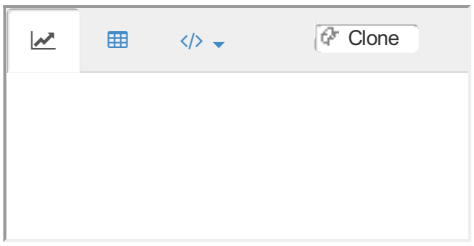
```
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Wikipedia Page Views data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Wikipedia Page Views dataset enables you to observe patterns in the traffic of company Wikipedia pages. Examples include the following strategies:

- Capitalizing on companies that have experienced a sharp increase in Wikipedia traffic on the premise that volatility in traffic will translate to volatility in price
- Mitigating risk by avoiding companies that have a decreasing web presence on the premise that a reduction in traffic will result in a reduction in price


Clone

Datasets

RegAlytics

RegAlytics was founded by Mary Kopczynski, Aaron Heisler, Alexander Appugliese, and Werner Pauliks in 2019 with the goal of significantly reducing the time and cost required to mitigate regulatory risk. RegAlytics provides access to accurate and clean regulatory data from all global regulators in all sectors that is enriched by regulatory experts for risk and compliance teams everywhere. Please come to RegAlytics directly if you would like data on other sectors or countries!

US Regulatory Alerts - Financial Sector

RegAlytics

US Regulatory Alerts - Financial Sector

Introduction

The US Regulatory Alerts dataset by RegAlytics tracks US regulatory changes within the financial services sector. The data covers 400,000 alerts, starts from January 2020, and is delivered on a daily basis. This dataset is created by sourcing information from over 5,000 regulators and using proprietary technology to gather and structure the regulatory data. Once prepared, the data is thoroughly reviewed by RegAlytics' team of regulatory experts and delivered each morning by 8AM for industry use.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Regulatory Alerts - Financial Sector dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

RegAlytics was founded by Mary Kopczynski, Aaron Heisler, Alexander Appugliese, and Werner Pauliks in 2019 with the goal of significantly reducing the time and cost required to mitigate regulatory risk. RegAlytics provides access to accurate and clean regulatory data from all global regulators in all sectors that is enriched by regulatory experts for risk and compliance teams everywhere. Please come to RegAlytics directly if you would like data on other sectors or countries!

Getting Started

The following snippet demonstrates how to request data from the US Regulatory Alerts - Financial Sector dataset:

```
PY
from QuantConnect.DataSource import *
self.dataset_symbol = self.AddData(RegalyticsRegulatoryArticles, "REG").Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2020
Coverage	400,000 Alerts
Data Density	Sparse
Resolution	Daily
Timezone	New York

Data Point Attributes

The US Regulatory Alerts dataset provides **RegalyticsRegulatoryArticle** and **RegalyticsRegulatoryArticles** objects.

RegalyticsRegulatoryArticle

RegalyticsRegulatoryArticle objects have the following attributes:

RegalyticsRegulatoryArticles

RegalyticsRegulatoryArticles objects have the following attributes:

Requesting Data

To add US Regulatory Alerts data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class RegalyticsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        self.dataset_symbol = self.AddData(RegalyticsRegulatoryArticles, "REG").Symbol
```

Accessing Data

To get the current US Regulatory Alerts data, index the current **Slice** with the dataset **Symbol** . **Slice** objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_points = slice[self.dataset_symbol]
        for data_point in data_points:
            self.Log(f"{self.dataset_symbol} title at {slice.Time}: {data_point.Title}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
def OnData(self, slice: Slice) -> None:
    data = slice.Get(RegalyticsRegulatoryArticles)
    if data:
        for articles in data.values():
            self.Log(f"{self.Time} {articles.ToString()}")

            for article in articles:
                self.Log(f"{self.dataset_symbol} article title at {slice.Time}: {article.Title}")
```

Historical Data

To get historical US Regulatory Alerts data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars =  self.History[RegalyticsRegulatoryArticles](self.dataset_symbol, 100, Resolution.Daily)
```

Remove Subscriptions

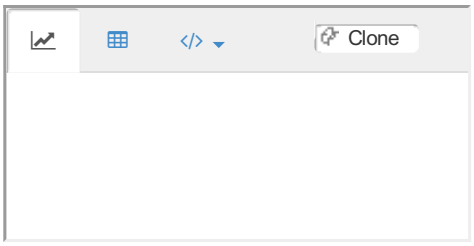
To remove your subscription to US Regulatory Alerts data, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The US Regulator Alters dataset enables you to accurately design strategies while mitigating regulatory risk. Examples include the following strategies:

- Temporarily increasing/decreasing exposure to securities when new regulations are announced
- Parsing the content of regulatory announcements to determine market or sector impact


Clone

Datasets

Securities and Exchange Commission

The mission of the U.S. Securities and Exchange Commission is to protect investors, maintain fair, orderly, and efficient markets, and facilitate capital formation. The SEC oversees the key participants in the securities world, including securities exchanges, securities brokers and dealers, investment advisors, and mutual funds. The SEC is concerned primarily with promoting the disclosure of important market-related information, maintaining fair dealing, and protecting against fraud.

US SEC Filings

Securities and Exchange Commission

US SEC Filings

Introduction

The US SEC Filings dataset provides the quarterly financial earning reports that the United States Securities and Exchange Commission (SEC) requires from publicly traded companies in the US. The data covers 15,000 US Equities, starts in January 1998, and is delivered on a daily frequency. The data is sourced from the SEC's Electronic Data Gathering, Analysis, and Retrieval (EDGAR) system. QuantConnect downloads and formats the Quarterly Financial Reports (10-Q) and Annual Financial Report (8-K) filings of companies into a format for easy consumption by LEAN.

For more information about the US SEC Filings dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The mission of the U.S. Securities and Exchange Commission is to protect investors, maintain fair, orderly, and efficient markets, and facilitate capital formation. The SEC oversees the key participants in the securities world, including securities exchanges, securities brokers and dealers, investment advisors, and mutual funds. The SEC is concerned primarily with promoting the disclosure of important market-related information, maintaining fair dealing, and protecting against fraud.

Getting Started

The following snippet demonstrates how to request data from the US SEC Filings dataset:

```py
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
self.report_8k_symbol = self.AddData(SECReport8K, self.symbol).Symbol
self.report_10k_symbol = self.AddData(SECReport10K, self.symbol).Symbol
self.report_10q_symbol = self.AddData(SECReport10Q, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	15,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Data Point Attributes

The US SEC Filings dataset provides **SECReport8K** , **SECReport10K** , and **SECReport10Q** objects.

Report 8K Attributes

SECReport8K objects have the following attributes:

Report 10K Attributes

SECReport10K objects have the following attributes:

Report 10Q Attributes

SECReport10Q objects have the following attributes:

Requesting Data

To add US SEC Filings data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class SECReportAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2019, 8, 21)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
        self.report_8k_symbol = self.AddData(SECReport8K, self.symbol).Symbol
        self.report_10k_symbol = self.AddData(SECReport10K, self.symbol).Symbol
        self.report_10q_symbol = self.AddData(SECReport10Q, self.symbol).Symbol
```

Accessing Data

To get the current US SEC Filings data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.report_8k_symbol):
        data_point = slice[self.report_8k_symbol]
        self.Log(f"{self.report_8k_symbol} report count at {slice.Time}:
{len(data_point.Report.Documents)}")

    if slice.ContainsKey(self.report_10k_symbol):
        data_point = slice[self.report_10k_symbol]
        self.Log(f"{self.report_10k_symbol} report count at {slice.Time}:
{len(data_point.Report.Documents)}")

    if slice.ContainsKey(self.report_10q_symbol):
        data_point = slice[self.report_10q_symbol]
        self.Log(f"{self.report_10q_symbol} report count at {slice.Time}:
{len(data_point.Report.Documents)}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(SECReport8K).items():
        self.Log(f"{dataset_symbol} report count at {slice.Time}: {len(data_point.Report.Documents)}")

    for dataset_symbol, data_point in slice.Get(SECReport10K).items():
        self.Log(f"{dataset_symbol} report count at {slice.Time}: {len(data_point.Report.Documents)}")

    for dataset_symbol, data_point in slice.Get(SECReport10Q).items():
        self.Log(f"{dataset_symbol} report count at {slice.Time}: {len(data_point.Report.Documents)}")
```

Historical Data

To get historical US SEC Filings data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
# DataFrames
report_8k_history_df = self.History(self.report_8k_symbol, 100, Resolution.Daily)
report_10k_history_df = self.History(self.report_10k_symbol, 100, Resolution.Daily)
report_10q_history_df = self.History(self.report_10q_symbol, 100, Resolution.Daily)
history_df = self.History([self.report_8k_symbol,
                           self.report_10k_symbol,
                           self.report_10q_symbol], 100, Resolution.Daily)

# Dataset objects
report_8k_history_bars = self.History[SECReport8K](self.report_8k_symbol, 100, Resolution.Daily)
report_10k_history_bars = self.History[SECReport10K](self.report_10k_symbol, 100, Resolution.Daily)
report_10q_history_bars = self.History[SECReport10Q](self.report_10q_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

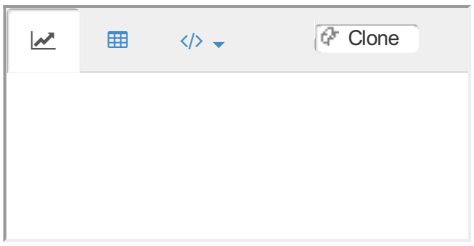
```
self.RemoveSecurity(self.report_8k_symbol)
self.RemoveSecurity(self.report_10k_symbol)
self.RemoveSecurity(self.report_10q_symbol)
```

If you subscribe to US SEC Filings data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The US SEC Filings dataset enables you to create strategies using information from SEC reports. Examples include the following strategies:

- Extracting information about corporate earnings from the documents for further analysis
- Applying sentiment analysis to the text content of the documents (for example, keyword scoring and ranking)


Clone

Datasets

Smart Insider

Smart Insider was founded by Michael Tindale in 2016 with the goal of forming the most progressive insider data vendor in the field. Smart Insider provides access to buyback intention and transactions for quantitative researchers. In addition to their Corporate Buybacks dataset, Smart Insider provides data on stock trades made by US politicians and thousands of high net worth individuals around the globe.

Corporate Buybacks

Smart Insider

Corporate Buybacks

Introduction

The Corporate Buybacks dataset by Smart Insider tracks US Equities share buyback programs. The data covers 3,000 US Equities, starts in May 2015, and is delivered on a second frequency. This dataset is created by analyzing daily buyback announcements and by using secondary data sources to ensure records are accurate and complete.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Corporate Buybacks dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Smart Insider was founded by Michael Tindale in 2016 with the goal of forming the most progressive insider data vendor in the field. Smart Insider provides access to buyback intention and transactions for quantitative researchers. In addition to their Corporate Buybacks dataset, Smart Insider provides data on stock trades made by US politicians and thousands of high net worth individuals around the globe.

Getting Started

The following snippet demonstrates how to request data from the Corporate Buybacks dataset:

```py
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
self.intention_symbol = self.AddData(SmartInsiderIntention, self.symbol).Symbol
self.transaction_symbol = self.AddData(SmartInsiderTransaction, self.symbol).Symbol

self.AddUniverse(SmartInsiderIntentionUniverse, "SmartInsiderIntentionUniverse", Resolution.Daily,
self.IntentionSelection)
self.AddUniverse(SmartInsiderTransactionUniverse, "SmartInsiderTransactionUniverse", Resolution.Daily,
self.TransactionSelection)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	May 2015
Asset Coverage	3,000 US Equities
Data Density	Sparse
Resolution	Second
Timezone	New York

Data Point Attributes

The Corporate Buybacks dataset provides **SmartInsiderIntention** , **SmartInsiderIntentionUniverse** , **SmartInsiderTransaction** , and **SmartInsiderTransactionUniverse** objects.

SmartInsiderIntention Attributes

SmartInsiderIntention objects have the following attributes:

SmartInsiderIntentionUniverse Attributes

SmartInsiderIntentionUniverse objects have the following attributes:

SmartInsiderTransaction Attributes

SmartInsiderTransaction objects have the following attributes:

SmartInsiderTransactionUniverse Attributes

SmartInsiderTransactionUniverse objects have the following attributes:

Requesting Data

To add Corporate Buybacks data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
class CorporateBuybacksDataAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2016, 1, 1)
        self.SetEndDate(2021, 1, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
        self.intention_symbol = self.AddData(SmartInsiderIntention, self.symbol).Symbol
        self.transaction_symbol = self.AddData(SmartInsiderTransaction, self.symbol).Symbol
```

Accessing Data

To get the current Corporate Buybacks data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.intention_symbol):
        data_point = slice[self.intention_symbol]
        self.Log(f"{self.intention_symbol} intention amount at {slice.Time}: {data_point.Amount}")

    if slice.ContainsKey(self.transaction_symbol):
        data_point = slice[self.transaction_symbol]
        self.Log(f"{self.transaction_symbol} transaction amount at {slice.Time}: {data_point.Amount}")
```

To iterate through all of the dataset objects in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, data_point in slice.Get(SmartInsiderIntention).items():
        self.Log(f"{dataset_symbol} intention amount at {slice.Time}: {data_point.Amount}")

    for dataset_symbol, data_point in slice.Get(SmartInsiderTransaction).items():
        self.Log(f"{dataset_symbol} transaction amount at {slice.Time}: {data_point.Amount}")
```

Historical Data

To get historical Corporate Buybacks data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrames
intention_history_df = self.History(self.intention_symbol, 100, Resolution.Daily)
transaction_history_df = self.History(self.transaction_symbol, 100, Resolution.Daily)
history_df = self.History([self.intention_symbol, self.transaction_symbol], 100, Resolution.Daily)

# Dataset objects
intention_history_bars = self.History[SmartInsiderIntention](self.intention_symbol, 100, Resolution.Daily)
transaction_history_bars = self.History[SmartInsiderTransaction](self.transaction_symbol, 100,
Resolution.Daily)
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Corporate Buybacks data, call the **AddUniverse** method with the **SmartInsiderIntentionUniverse** class or the **SmartInsiderTransactionUniverse** and a selection function.

```
PY
def Initialize(self) -> None:
    self.AddUniverse(SmartInsiderIntentionUniverse, "SmartInsiderIntentionUniverse", Resolution.Daily,
self.IntentionSelection)
    self.AddUniverse(SmartInsiderTransactionUniverse, "SmartInsiderTransactionUniverse", Resolution.Daily,
self.TransactionSelection)

def IntentionSelection(self, alt_coarse: List[SmartInsiderIntentionUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.Percentage > 0.005 \
                and d.USDMarketCap > 100000000]

def TransactionSelection(self, alt_coarse: List[SmartInsiderTransactionUniverse]) -> List[Symbol]:
    return [d.Symbol for d in alt_coarse \
                if d.BuybackPercentage > 0.005 \
                and d.USDMarketCap > 100000000]
```

For more information about dynamic universes, see Universes .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

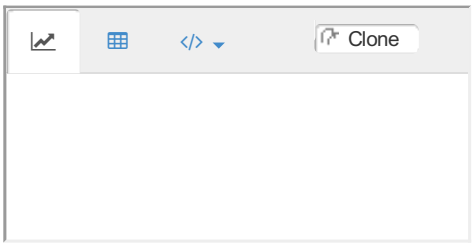
```
PY
self.RemoveSecurity(self.intention_symbol)
self.RemoveSecurity(self.transaction_symbol)
```

If you subscribe to Corporate Buybacks data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Corporate Buybacks dataset enables you to design strategies using information on company buyback programs. Examples include the following strategies:

- Buying securities when the company announces an upcoming share buyback on the premise that the reduction in supply (shares outstanding) will drive up the remaining shares price
- Buying securities when the company executes an upcoming share buyback on the premise that the reduction in supply (shares outstanding) will drive up the remaining shares price



Datasets

Tiingo

Tiingo was founded by Rishi Singh in 2014. Tiingo goes beyond traditional news sources and focuses on finding rich, quality content written by knowledgeable writers. Their proprietary algorithms scan unstructured, non-traditional news and other information sources while tagging companies, topics, and assets. This refined system is backed by over ten years of research and development, and is written by former institutional quant traders. Because of this dedicated approach, Tiingo"s News API is a trusted tool used by quant funds, hedge funds, pension funds, social media companies, and tech companies around the world.

Tiingo News Feed

Tiingo

Tiingo News Feed

Introduction

The Tiingo News Feed dataset by Tiingo tracks US Equity news releases. The data covers 10,000 US Equities, starts in January 2014, and is delivered on a second frequency. This dataset is creating by Tiingo integrating over 120 different news providers into their platform.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Tiingo News Feed dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Tiingo was founded by Rishi Singh in 2014. Tiingo goes beyond traditional news sources and focuses on finding rich, quality content written by knowledgeable writers. Their proprietary algorithms scan unstructured, non-traditional news and other information sources while tagging companies, topics, and assets. This refined system is backed by over ten years of research and development, and is written by former institutional quant traders. Because of this dedicated approach, Tiingo's News API is a trusted tool used by quant funds, hedge funds, pension funds, social media companies, and tech companies around the world.

Getting Started

The following snippet demonstrates how to request data from the Tiingo News Feed dataset:

```
PY
from QuantConnect.DataSource import *

self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
self.dataset_symbol = self.AddData(TiingoNews, self.symbol).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2014
Asset Coverage	10,000 US Equities
Data Density	Sparse
Resolution	Second
Timezone	UTC

Data Point Attributes

The Tiingo News Feed dataset provides **TiingoNews** objects, which have the following attributes:

Requesting Data

To add Tiingo News Feed data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class TiingoNewsDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2021, 6, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("AAPL", Resolution.Minute).Symbol
        self.dataset_symbol = self.AddData(TiingoNews, self.symbol).Symbol
```

Accessing Data

To get the current Tiingo News Feed data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        article = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} article description at {slice.Time}: {article.Description}")
```

To iterate through all of the articles in the current **Slice** , call the **Get** method.

```
PY
def OnData(self, slice: Slice) -> None:
    for dataset_symbol, article in slice.Get(TiingoNews).items():
        self.Log(f"{dataset_symbol} article description at {slice.Time}: {article.Description}")
```

Historical Data

To get historical Tiingo News Feed data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```py
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
self.History[TiingoNews](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the **RemoveSecurity** method.

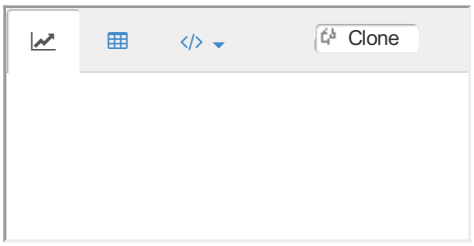
```py
self.RemoveSecurity(self.dataset_symbol)
```

If you subscribe to Tiingo News Feed data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Tiingo News Feed enables you to accurately design strategies harnessing news articles on the companies you're trading. Examples include the following strategies:

- Creating a dictionary of sentiment scores for various words and assigning a sentiment score to the content of each news release
- Calculating the sentiment of news releases with Natural Language Processing (NLP)
- Trading securities when their news releases are tagged by Tiingo with current buzzwords



Datasets

Treasury Department

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation"s financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

US Treasury Yield Curve

Treasury Department

US Treasury Yield Curve

Introduction

The US Treasury Yield Curve datasets tracks the yield curve rate from the US Department of the Treasury. The data starts in January 1990 and is delivered on a daily frequency. This dataset is calculated from composites of indicative, bid-side market quotations (not actual transactions) obtained by the Federal Reserve Bank of New York at or near 3:30 PM Eastern Time (ET) each trading day.

For more information about the US Treasury Yield Curve dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation's financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

Getting Started

The following snippet demonstrates how to request data from the US Treasury Yield Curve dataset:

```
PY
from QuantConnect.DataSource import *
self.dataset_symbol = self.AddData(USTreasuryYieldCurveRate, "USTYCR").Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1990
Coverage	1 Dataset
Data Density	Sparse
Resolution	Daily
Timezone	New York

Data Point Attributes

The US Treasury Yield Curve dataset provides **USTreasuryYieldCurveRate** objects, which have the following attributes:

Requesting Data

To add US Treasury Yield Curve data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY

class QuiverCongressDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2019, 1, 1)
        self.SetEndDate(2020, 6, 1)
        self.SetCash(100000)

        self.dataset_symbol = self.AddData(USTreasuryYieldCurveRate, "USTYCR").Symbol
```

Accessing Data

To get the current US Treasury Yield Curve data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY

def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} one month value at {slice.Time}: {data_point.OneMonth}")
```

Historical Data

To get historical US Treasury Yield Curve data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY

# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[USTreasuryYieldCurveRate](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove your subscription to US Treasury Yield Curve data, call the **RemoveSecurity** method.

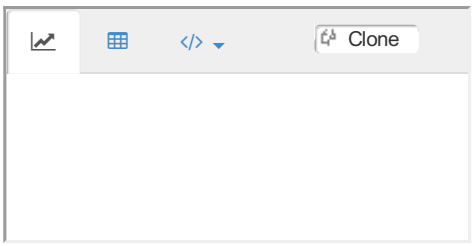
```
PY

self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The US Treasury Yield Curve dataset enables you to monitor the yields of bonds with numerous maturities in your

strategies. Examples include the following strategies:

- Short selling SPY when the yield curve inverts
- Buying short-term Treasuries and short selling long-term Treasuries when the yield curve becomes steeper (aka curve steepener trade)



Datasets

VIX Central

VIX Central was founded by Eli Mintz in 2012 with goal of displaying historical VIX term structures in a simple and intuitive interface. VIX Central provides access to real-time and historical VIX data for individual investors.

VIX Central Contango

VIX Central

VIX Central Contango

Introduction

The VIX Central Contango dataset by VIX Central tracks VIX Futures (VX) contango data. The data covers 12 Futures contracts closest to expiry/maturity, starts in June 2010, and is delivered on a daily frequency. The dataset is created by QuantConnect downloading data from VIX Central website, which collects and analyses VIX and VX (VIX Futures) data.

Contango and Backwardation are terms used to describe if participants in the Futures market are overpaying or underpaying relative to the "spot" price of the underlying commodity when trading a Futures contract ("spot" price is the price of the actual commodity/asset at a given moment in time). Contango and backwardation can be used to determine forward-looking expectations of the commodity's spot price by the time the Future has expired/matured and is set to be delivered by participants of the Futures market. As Futures near their expiration/maturity date, contango and backwardation curves tend to converge on the spot price of the commodity at the time of expiration.

For more information about the VIX Central Contango dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

VIX Central was founded by Eli Mintz in 2012 with goal of displaying historical VIX term structures in a simple and intuitive interface. VIX Central provides access to real-time and historical VIX data for individual investors.

Getting Started

The following snippet demonstrates how to request data from the VIX Central Contango dataset:

```
PY
from QuantConnect.DataSource import *
self.dataset_symbol = self.AddData(VIXCentralContango, "VIX", Resolution.Daily).Symbol
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	June 2010
Asset Coverage	1 Futures Chain with 12 contracts
Data Density	Regular
Resolution	Daily
Timezone	New York

Data Point Attributes

The VIX Central Contango dataset provides **VIXCentralContango** objects, which have the following attributes:

Requesting Data

To add VIX Central Contango data to your algorithm, call the **AddData** method. Save a reference to the dataset **Symbol** so you can access the data later in your algorithm.

```
PY
class VixCentralContangoAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2014, 1, 1)
        self.SetEndDate(2018, 1, 1)
        self.SetCash(25000)

        self.dataset_symbol = self.AddData(VIXCentralContango, "VX", Resolution.Daily).Symbol
```

Accessing Data

To get the current VIX Central Contango data, index the current **Slice** with the dataset **Symbol** . Slice objects deliver unique events to your algorithm as they happen, but the **Slice** may not contain data for your dataset at every time step. To avoid issues, check if the **Slice** contains the data you want before you index it.

```
PY
def OnData(self, slice: Slice) -> None:
    if slice.ContainsKey(self.dataset_symbol):
        data_point = slice[self.dataset_symbol]
        self.Log(f"{self.dataset_symbol} front month at {slice.Time}: {data_point.FrontMonth}")
```

Historical Data

To get historical VIX Central Contango data, call the **History** method with the dataset **Symbol** . If there is no data in the period you request, the history result is empty.

```
PY
# DataFrame
history_df = self.History(self.dataset_symbol, 100, Resolution.Daily)

# Dataset objects
history_bars = self.History[VIXCentralContango](self.dataset_symbol, 100, Resolution.Daily)
```

For more information about historical data, see History Requests .

Remove Subscriptions

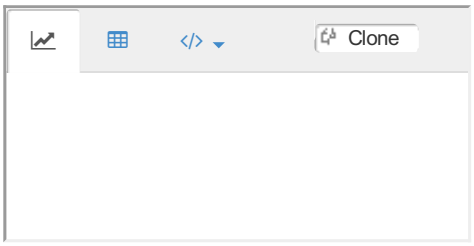
To remove your subscription to VIX Central Contango data, call the **RemoveSecurity** method.

```
self.RemoveSecurity(self.dataset_symbol)
```

Example Applications

The VIX Central Contango dataset enable you to explore VIX Future contracts pricing data. Examples include the following strategies:

- Determining forward-looking expectations by Futures market participants of the underlying commodity's spot price by the time of expiration/maturity
- Creating cash-and-carry arbitrage strategies by trading the spread/convergence of the contango/backwardation curves as a Future nears expiration/maturity



Importing Data

Importing Data

Key Concepts

Introduction

Custom data is your own external data that's not from the Dataset Market. You can use custom data to inform trading decisions and to simulate trades on unsupported securities. To get custom data into your algorithms, you download the entire file at once or read it line-by-line with a custom data reader. If you use a custom data reader, LEAN sends the data to the `OnData` method in your algorithm.

Stream Custom Data

To receive your custom data in the `OnData` method, create a custom type and then create a data subscription. The custom data type tells LEAN where to get your data and how to read it.

All custom data types must extend the `PythonData` class and override the `GetSource` and `Reader` methods

```py
class MyCustomDataType(PythonData):
    def GetSource(self,
                  config: SubscriptionDataConfig,
                  date: datetime,
                  isLive: bool) -> SubscriptionDataSource:
        return SubscriptionDataSource("<sourceURL>", SubscriptionTransportMedium.RemoteFile)

    def Reader(self,
                  config: SubscriptionDataConfig,
                  line: str,
                  date: datetime,
                  isLive: bool) -> BaseData:

                  if not (line.strip() and line[0].isdigit()):
                     return None

                  data = line.split(',')

        custom = MyCustomDataType()
        custom.Time = datetime.strptime(data[0], '%Y%m%d')
        custom.EndTime = custom.Time + timedelta(1)
        custom.Value = float(data[1])
        custom["Property1"] = float(data[2])
        return custom
```

For more information about custom data types, see Streaming Data .

Download Bulk Data

The `Download` method downloads the content served from a local file or URL and then returns it as a string.

```
content = self.Download("<filePathOrURL>")
```

For more information about bulk downloads, see Bulk Downloads .

Remote File Providers

The most common remote file providers to use are Dropbox, GitHub, and Google Sheets.

Dropbox

If you store your custom data in Dropbox, you need to create a link to the file and add `?dl=1` to the end of the file URL. To create file links, see How to share files or folders in the Dropbox documentation. The `?dl=1` parameter lets you download the direct file link, not the HTML page of the file.

GitHub

If you store your custom data in GitHub, you can use public or private repositories. When you download the data, use the raw file (for example, **https://raw.githubusercontent.com/<organization>/<repo>/<path>**). For instructions on accessing the raw file, see Viewing or copying the raw file content in the GitHub documentation.

Google Sheets

If you store your custom data in Google Sheets, you need to create a link to the file. To create file links, see Make Google Docs, Sheets, Slides & Forms public in the Google Docs documentation. Choose the **CSV** format to find `/pub?gid={SHEET_ID}&single=true&output=csv` parameter on the end of the link. Google Sheets don't support exporting data in **JSON** format.

File Quotas

The following table shows the number of files you can download during a single backtest or Research Environment session in QuantConnect Cloud:

Organization Tier	File Quota
Free	25
Quant Researcher	100
Team	250
Trading Firm	1,000
Free	Unlimited

Each file can be up to 200 MB in size and have a file name up to 200 characters long.

If you need to import more files than your quota allows, save your custom data files in the Object Store and load them from there.

Rate Limits

We do not impose a rate limit on file downloads but often external providers do. Dropbox caps download speeds to 10 kb/s after 3-4 download requests. To ensure your algorithms run fast, only use a small number of small custom data files.

Timeouts

In cloud algorithms, the download methods have a 10-second timeout period. If the methods don't download the data within 10 seconds, LEAN throws an error.

Importing Data

Streaming Data

Streaming Data

Key Concepts

Introduction

There are two techniques to import data into your algorithm. You can either manually import the entire file or stream the file line-by-line into your algorithm's `OnData` event. This page explores streaming a file's contents into your algorithm line-by-line.

Data Formats

Common data formats are **CSV** , **JSON** , and **XML** , but you can use any file type that can be read over the internet. Each request has a one-second overhead, so you should package your custom data to minimize requests. Bundle dates together where possible to speed up execution. Just ensure the data in the file is in chronological order.

For Excel files, please double check the raw data format for parsing in the data reader, since data will be formatted for convenient visualization in Excel application view. To avoid confusion of data format, save the spreadsheet as a **CSV** file and open it in a text editor to confirm the raw data format.

Set Data Sources

The `GetSource` method in your custom data class instructs LEAN where to find the data. This method must return a `SubscriptionDataSource` object, which contains the data location and format.

The following table describes the arguments the `SubscriptionDataSource` accepts:

Argument	Data Type	Description	Default Value
source	str	Data source location	
transportMedium	SubscriptionTransportMedium	The transport medium to be used to retrieve data from the source	
format	FileFormat	The format of the data within the source	FileFormat.Csv
headers	IEnumerable<KeyValuePair<string, string>>	The headers to be used for this source. In cloud algorithms, each of the key-value pairs can consist of up to 1,000 characters.	None

The SubscriptionTransportMedium enumeration has the following members:

The FileFormat enumeration has the following members:

The following table describes the arguments the GetSource method accepts:

Argument	Data Type	Description
config	SubscriptionDataConfig	The subscription configuration
date	datetime	Date of this source file
isLiveMode	bool	True if algorithm is running in live mode

You can use these arguments to create SubscriptionDataSource objects representing different locations and formats.

```
class MyCustomDataType(PythonData):
    def GetSource(self,
         config: SubscriptionDataConfig,
         date: datetime,
         isLiveMode: bool) -> SubscriptionDataSource:

         if isLiveMode:
            return SubscriptionDataSource("https://www.bitstamp.net/api/ticker/",
SubscriptionTransportMedium.Rest)

        source = f"http://my-ftp-server.com/{config.Symbol.Value}/{date:%Y%M%d}.csv"
        return SubscriptionDataSource(source, SubscriptionTransportMedium.RemoteFile)
```

Parse Custom Data

The Reader method of your custom data class takes one line of data from the source location and parses it into one of your custom objects. You can add as many properties to your custom data objects as you need, but the following table describes the properties you must set. When there is no useable data in a line, the method should return None . LEAN repeatedly calls the Reader method until the date/time advances or it reaches the end of the file.

Property	Description
`Symbol`	You can set this property to `config.Symbol` .
`Time`	The time when the data sample starts.
`EndTime`	The time when the data sample ends and when LEAN should add the sample to a Slice .
`Value`	The default data point value.

The following table describes the arguments the `Reader` method accepts:

Argument	Data Type	Description
`config`	`SubscriptionDataConfig`	The subscription configuration
`line`	`str`	Content from the requested data source
`date`	`datetime`	Date of this source file
`isLiveMode`	`bool`	`True` if algorithm is running in live mode

You can use these arguments to create `BaseData` objects from different sources.

```
class MyCustomDataType(PythonData):
    def Reader(self,
         config: SubscriptionDataConfig,
         line: str,
         date: datetime,
         isLiveMode: bool) -> BaseData:

        if not line.strip():
            return None

        custom = MyCustomDataType()
        custom.Symbol = config.Symbol

        if isLiveMode:
            data = json.loads(line)
            custom.EndTime =  Extensions.ConvertFromUtc(datetime.utcnow(), config.ExchangeTimeZone)
            custom.Value = data["value"]
            return custom

        if not line[0].isdigit():
            return None

        data = line.split(',')
        custom.EndTime = datetime.strptime(data[0], '%Y%m%d') + timedelta(1)
        custom.Value = float(data[1])
        return custom
```

Demonstration Algorithm

The following example algorithm implements a custom data source for the Bitstamp API.

```
class CustomDataBitstampAlgorithm(QCAlgorithm):
```

```
    def Initialize(self):
        self.SetStartDate(2012, 9, 13)
        self.SetEndDate(2021, 6, 20)
        self.SetCash(100000)

        # Define the symbol and "type" of our generic data:
        self.custom_data_symbol = self.AddData(Bitstamp, "BTC").Symbol

        history = self.History(Bitstamp, self.custom_data_symbol, 200, Resolution.Daily)
        self.Debug(f"We got {len(history)} items from historical data request of
{self.custom_data_symbol}.")


    def OnData(self, slice):
        if self.custom_data_symbol not in slice:
            return

        data = slice[self.custom_data_symbol]
        self.Log(f'{data.EndTime}: Close: {data.Close}')
        self.Plot(self.custom_data_symbol, 'Price', data.Close)


class Bitstamp(PythonData):

    def GetSource(self, config, date, isLiveMode):
        if isLiveMode:
            return SubscriptionDataSource("https://www.bitstamp.net/api/ticker/",
SubscriptionTransportMedium.Rest)

        source =
"https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/bitstampusd.csv"
        return SubscriptionDataSource(source, SubscriptionTransportMedium.RemoteFile)

    def Reader(self, config, line, date, isLiveMode):

        if not line.strip():
            return None

        coin = Bitstamp()
        coin.Symbol = config.Symbol

        if isLiveMode:
            # Example Line Format:
            # {"high": "441.00", "last": "421.86", "timestamp": "1411606877", "bid": "421.96", "vwap":
"428.58", "volume": "14120.40683975", "low": "418.83", "ask": "421.99"}
            liveBTC = json.loads(line)

            # If value is zero, return None
            coin.Value = float(liveBTC["last"])
            if coin.Value == 0:
                return None

            coin.EndTime =  Extensions.ConvertFromUtc(datetime.utcnow(), config.ExchangeTimeZone)
            coin.Time = coin.EndTime - timedelta(1)
            coin["Open"] = float(liveBTC["open"])
            coin["High"] = float(liveBTC["high"])
            coin["Low"] = float(liveBTC["low"])
            coin["Close"] = coin.Value
            coin["Ask"] = float(liveBTC["ask"])
            coin["Bid"] = float(liveBTC["bid"])
            coin["VolumeBTC"] = float(liveBTC["volume"])
            coin["WeightedPrice"] = float(liveBTC["vwap"])
            return coin

        # Example Line Format:
        # Date      Open   High    Low     Close   Volume (BTC)    Volume (Currency)   Weighted Price
        # 2011-09-13 5.8    6.0     5.65    5.97    58.37138238,    346.0973893944      5.929230648356
        if not line[0].isdigit():
            return None

        data = line.split(',')

        # If value is zero, return None
        coin.Value = float(data[4])
        if coin.Value == 0:
            return None

        coin.Time = datetime.strptime(data[0], "%Y-%m-%d")
        coin.EndTime = coin.Time + timedelta(1)
```

```
        coin.EndTime = coin.Time + timedelta(1)
        coin["Open"] = float(data[1])
        coin["High"] = float(data[2])
        coin["Low"] = float(data[3])
        coin["Close"] = coin.Value
        coin["VolumeBTC"] = float(data[5])
        coin["VolumeUSD"] = float(data[6])
        coin["WeightedPrice"] = float(data[7])
        return coin
```

To save this algorithm to your cloud projects, clone it .

Streaming Data

Custom Securities

Custom Securities

Key Concepts

Introduction

To receive your custom data in the `OnData` method instead of in a bulk download, create a custom type and then create a data subscription. The custom data type tells LEAN where to get your data and how to read it. All custom data types must extend from `PythonData` and override the `Reader` and `GetSource` methods.

Unlike custom data, native QC data gets the full benefit of security modeling like splits and dividends. Custom data is ignorant of everything about the actual market. Things like market hours, delistings, and mergers don't work with custom data.

Create Subscriptions

After you define the custom data class, in the `Initialize` method of your algorithm, call the `self.AddData(Type class, string ticker, Resolution resolution = Resolution.Daily)` method. This method gives LEAN the type factory to create the data objects, the name of the data, and the resolution to poll the data source for updates.

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddData(MyCustomDataType, "<name>", Resolution.Daily).Symbol
```

The `resolution` argument should match the resolution of your custom dataset. The lowest reasonable resolution is every minute. Anything more frequent than every minute is very slow to execute. The frequency that LEAN checks the data source depends on the `resolution` argument. The following table shows the polling frequency of each resolution:

Resolution	Update Frequency
Daily	Every 30 minutes
Hour	Every 30 minutes
Minute	Every minute
Second	Every second
Tick	Constantly checks for new data

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points in the `Slice` it passes to your algorithm's `OnData` method. To collect the custom data, use the `Symbol` or name of your custom data subscription. You can access the `Value` and custom properties of your custom data class from the `Slice` . To access the custom properties, pass the property name to the `GetProperty` method .

```
PY
class MyAlgorithm(QCAlgorithm):
    def OnData(self, slice: Slice) -> None:
        if slice.ContainsKey(self.symbol):
            custom_data = slice[self.symbol]
            value = custom_data.Value
            property1 = custom_data.Property1
```

Set the Benchmark

To set your custom data source as the benchmark, in the `Initialize` method, call the `SetBenchmark` method with the `Symbol` of your custom data subscription.

```
PY
self.symbol = self.AddData(MyCustomDataType, "<name>", Resolution.Daily).Symbol
self.SetBenchmark(self.symbol)
```

Avoid Look-Ahead Bias

Look-ahead bias occurs when you make decisions with information that wouldn't be available until some time in the future. In backtesting, look-ahead bias occurs when you receive data earlier than it would actually be available in reality. If look-ahead bias seeps into your algorithm, it will perform differently between live trading and backtesting.

To avoid look-ahead bias, set the timestamp of data points to the time when the data would actually be available. A lot of external sources apply timestamps to data differently than we do. For instance, on some platforms, daily bars are displayed with the date that they opened. We display daily bars with the date they closed. If you set the `EndTime` to the start of the bar, you'll receive the bar earlier in backtests than you would in live trading.

Time Modeling

Data types classes in LEAN inherit from the `BaseData` class that defines the `Time` and `EndTime` properties. These properties represent the time period of which the data was built. If the data type occurs in a singular point in time, they

have no period, so `Time` and `EndTime` are the same. Regardless of the period, LEAN uses the time when the data sample ends, `EndTime` , to add the sample to a `Slice` .

Custom Securities

CSV Format Example

Introduction

This page explains how to import custom data of a single security sourced in **CSV** format.

Data Format

The following snippet shows an example **CSV** file:

```
Date,Open,High,Low,Close,SharesTraded,Tur11er(Rs.Cr)
1997-01-01,905.2,941.4,905.2,939.55,38948210,978.21
1997-01-02,941.95,944,925.05,927.05,49118380,1150.42
1997-01-03,924.3,932.6,919.55,931.65,35263845,866.74
...
2014-07-24,7796.25,7835.65,7771.65,7830.6,117608370,6271.45
2014-07-25,7828.2,7840.95,7748.6,7790.45,153936037,7827.61
2014-07-28,7792.9,7799.9,7722.65,7748.7,116534670,6107.78
```

The data in the file must be in chronological order.

Define Custom Types

To define a custom data type, inherit the `PythonData` class and override the GetSource and Reader methods.

```
PY
class MyCustomDataType(PythonData):

    def GetSource(self,
         config: SubscriptionDataConfig,
         date: datetime,
         isLive: bool) -> SubscriptionDataSource:
        return SubscriptionDataSource("https://www.dropbox.com/s/rsmg44jr6wexn2h/CNXNIFTY.csv?dl=1",
SubscriptionTransportMedium.RemoteFile)

    def Reader(self,
         config: SubscriptionDataConfig,
         line: str,
         date: datetime,
         isLive: bool) -> BaseData:

        if not (line.strip()):
            return None

        index = MyCustomDataType()
        index.Symbol = config.Symbol

        try:
            data = line.split(',')
            index.Time = datetime.strptime(data[0], "%Y-%m-%d")
            index.EndTime = index.Time + timedelta(days=1)
            index.Value = data[4]
            index["open"] = float(data[1])
            index["high"] = float(data[2])
            index["low"] = float(data[3])
            index["close"] = float(data[4])

        except ValueError:
            # Do nothing
            return None

        return index
```

Create Subscriptions

To create a subscription for the custom type, in the `Initialize` method, call the `AddData` method. The `AddData` method returns a `Security` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
PY
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddData(MyCustomDataType, "MyCustomDataType", Resolution.Daily).Symbol
```

The `AddData` method creates a subscription for a single custom data asset and adds it to your **user-defined** universe.

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points in the `Slice` it passes to your algorithm's `OnData` method. To collect the custom data, use the `Symbol` or name of your custom data subscription. You can access the `Value` and custom properties of your custom data class from the `Slice` . To access the custom properties, pass the property name to the `GetProperty` method .

```
class MyAlgorithm(QCAlgorithm):
    def OnData(self, slice: Slice) -> None:
        if slice.ContainsKey(self.symbol):
            custom_data = slice[self.symbol]
            close = custom_data.Close
```

If you add custom properties to your data object in the `Reader` method, LEAN adds them as members to the data object in the `Slice` . To ensure the property names you add in the `Reader` method follow the convention of member names, LEAN applies the following changes to the property names you provide in the `Reader` method:

1. `-` and `.` characters are replaced with whitespace.
2. The first letter is capitalized.
3. Whitespace characters are removed.

For example, if you set a property name in the `Reader` method to `['some-property.name']` , you can access it in the `OnData` method through the `Somepropertyname` member of your data object.

Demonstration Algorithms

BubbleAlgorithm.py Python CustomDataNIFTYAlgorithm.py Python

Custom Securities

JSON Format Example

Introduction

This page explains how to import custom data of a single security sourced in **JSON** format.

Data Format

The following code snippet shows an example **JSON** file

```
[
    {
      "Date": "1997-01-01",
      "Open": 905.2,
      "High": 941.4,
      "Low": 905.2,
      "Close": 939.55,
      "SharesTraded": 38948210,
      "Tur11er(Rs.Cr)": 978.21
    },
    {
      "Date": "1997-01-02",
      "Open": 941.95,
      "High": 944,
      "Low": 925.05,
      "Close": 927.05,
      "SharesTraded": 49118380,
      "Tur11er(Rs.Cr)": 1150.42
    },
    ...
    {
      "Date": "2014-07-25",
      "Open": 7828.2,
      "High": 7840.95,
      "Low": 7748.6,
      "Close": 7790.45,
      "SharesTraded": 153936037,
      "Tur11er(Rs.Cr)": 7827.61
    },
    {
      "Date": "2014-07-28",
      "Open": 7792.9,
      "High": 7799.9,
      "Low": 7722.65,
      "Close": 7748.7,
      "SharesTraded": 116534670,
      "Tur11er(Rs.Cr)": 6107.78
    }
]
```

The data in the file must be in chronological order.

Define Custom Types

To define a custom data type, inherit the `PythonData` class and override the GetSource and Reader methods.

If you need to create multiple objects in your `Reader` method from a single `line` , follow these steps:

1. In the `GetSource` method, pass `FileFormat.UnfoldingCollection` as the third argument to the

`SubscriptionDataSource` constructor.

2. In the `Reader` method, order the objects by their timestamp and then return a `BaseDataCollection(endTime, config.Symbol, objects)` where `objects` is a list of your custom data objects.

```
class MyCustomDataType(PythonData):
    def GetSource(self,
         config: SubscriptionDataConfig,
         date: datetime,
         isLive: bool) -> SubscriptionDataSource:
        return
SubscriptionDataSource("https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/
data/unfolding-collection-example.json", SubscriptionTransportMedium.RemoteFile,
FileFormat.UnfoldingCollection)

    def Reader(self,
         config: SubscriptionDataConfig,
         line: str,
         date: datetime,
         isLive: bool) -> BaseData:

        if not (line.strip()):
            return None

        objects = []
        data = json.loads(line)

        for datum in data:
            index = MyCustomDataType()
            index.Symbol = config.Symbol
            index.Time = datetime.strptime(datum["Date"], "%Y-%m-%d")
            index.EndTime = index.Time + timedelta(1)
            index["Open"] = float(datum["Open"])
            index["High"] = float(datum["High"])
            index["Low"] = float(datum["Low"])
            index["Close"] = float(datum["Close"])
            index.Value = index["Close"]
            objects.append(index)

        return BaseDataCollection(objects[-1].EndTime, config.Symbol, objects)
```

Create Subscriptions

To create a subscription for the custom type, in the `Initialize` method, call the `AddData` method. The `AddData` method returns a `Security` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddData(MyCustomDataType, "MyCustomDataType", Resolution.Daily).Symbol
```

The `AddData` method creates a subscription for a single custom data asset and adds it to your **user-defined** universe.

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points in the `Slice` it passes to your algorithm's `OnData` method. To collect the custom data, use the `Symbol` or name of your custom data subscription. You can access the `Value` and custom properties of your custom data class from the `Slice` . To access the custom properties, pass the property name to the `GetProperty` method .

```
class MyAlgorithm(QCAlgorithm):
    def OnData(self, slice: Slice) -> None:
        if slice.ContainsKey(self.symbol):
            custom_data = slice[self.symbol]
            close = custom_data.Close
```

If you add custom properties to your data object in the `Reader` method, LEAN adds them as members to the data object in the `Slice` . To ensure the property names you add in the `Reader` method follow the convention of member names, LEAN applies the following changes to the property names you provide in the `Reader` method:

1. `-` and `.` characters are replaced with whitespace.
2. The first letter is capitalized.
3. Whitespace characters are removed.

For example, if you set a property name in the `Reader` method to `['some-property.name']` , you can access it in the `OnData` method through the `Somepropertyname` member of your data object.

Demonstration Algorithms

CustomDataBitcoinAlgorithm.py Python

Streaming Data

Custom Universes

Custom Universes

Key Concepts

Introduction

A custom universe lets you select a basket of assets from a custom dataset.

Initialize Universes

To add a custom universe to your algorithm, in the `Initialize` method, pass your universe type and a selector function to the `AddUniverse` method.

```
PY
self.AddUniverse(MyCustomUniverseDataClass, "myCustomUniverse", Resolution.Daily, self.selector_function)
```

Receive Custom Data

The universe selector function receives a list of your custom objects and must return a list of `Symbol` objects. In the selector function definition, you can use any of the properties of your custom data type. The `Symbol` objects that you return from the selector function set the constituents of the universe.

```
PY
class MyCustomUniverseAlgorithm(QCAlgorithm):
 def selector_function(self, data: List[MyCustomUniverseDataClass]) -> List[Symbol]:
     sorted_data = sorted([ x for x in data if x["CustomAttribute1"] > 0 ],
                          key=lambda x: x["CustomAttribute2"],
                          reverse=True)
     return [x.Symbol for x in sorted_data[:5]]
```

Custom Universes

CSV Format Example

Introduction

This page explains how to import custom data for universe selection sourced in **CSV** format.

Data Format

You must create a file with data in **CSV** format. Ensure the data in the file is in chronological order.

```
20170704,SPY,QQQ,FB,AAPL,IWM
20170706,QQQ,AAPL,IWM,FB,GOOGL
20170707,IWM,AAPL,FB,BAC,GOOGL
...
20170729,SPY,QQQ,FB,AAPL,IWM
20170801,QQQ,FB,AAPL,IWM,GOOGL
20170802,QQQ,IWM,FB,BAC,GOOGL
```

Define Custom Types

To define a custom data type, inherit the `PythonData` class and override the GetSource and Reader methods.

PY

```
class StockDataSource(PythonData):

    def GetSource(self,
                  config: SubscriptionDataConfig,
                  date: datetime,
                  isLive: bool) -> SubscriptionDataSource:
        return SubscriptionDataSource("https://www.dropbox.com/s/ae1couew5ir3z9y/daily-stock-picker-
backtest.csv?dl=1", SubscriptionTransportMedium.RemoteFile)

    def Reader(self,
                  config: SubscriptionDataConfig,
                  line: str,
                  date: datetime,
                  isLive: bool) -> BaseData:

        if not (line.strip() and line[0].isdigit()): return None

        stocks = StockDataSource()
        stocks.Symbol = config.Symbol

        try:
            csv = line.split(',')
            stocks.Time = datetime.strptime(csv[0], "%Y%m%d")
            stocks.EndTime = stocks.Time + timedelta(days=1)
            stocks["Symbols"] = csv[1:]

        except ValueError:
            # Do nothing
            return None

        return stocks
```

Initialize Universe

To perform a universe selection with custom data, in the `Initialize` method, call the `AddUniverse` method.

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.AddUniverse(StockDataSource, "my-stock-data-source", Resolution.Daily, self.FilterFunction)
```

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points into a `List[StockDataSource])` object it passes to your algorithm's filter function. Your filter function needs to return a list of `Symbol` or `str` object. LEAN automatically subscribes to these new assets and adds them to your algorithm.

```
class MyAlgorithm(QCAlgorithm):
    def FilterFunction(self, data: List[StockDataSource]) -> List[str]:
        list = []
        for item in data:
            for symbol in item["Symbols"]:
                list.append(symbol)
        return list
```

If you add custom properties to your data object in the `Reader` method, LEAN adds them as members to the data object in your filter method. To ensure the property names you add in the `Reader` method follow the convention of member names, LEAN applies the following changes to the property names you provide in the `Reader` method:

1. `-` and `.` characters are replaced with whitespace.
2. The first letter is capitalized.
3. Whitespace characters are removed.

For example, if you set a property name in the `Reader` method to `['some-property.name']` , you can access it in your filter method through the `Somepropertyname` member of your data object.

Demonstration Algorithms

DropboxBaseDataUniverseSelectionAlgorithm.py Python

Custom Universes

JSON Format Example

Introduction

This page explains how to import custom data for universe selection sourced in **JSON** format.

Data Format

You must create a file with data in **JSON** format. Ensure the data in the file is in chronological order.

```
[
    {
      "Date": "20170704",
      "Symbols": ["SPY", "QQQ", "FB", "AAPL", "IWM"]
    },
    {
      "Date": "20170706",
      "Symbols": ["QQQ", "AAPL", "IWM", "FB", "GOOGL"]
    },
    ...
    {
      "Date": "20170801",
      "Symbols": ["QQQ", "FB", "AAPL", "IWM", "GOOGL"]
    },
    {
      "Date": "20170802",
      "Symbols": ["QQQ", "IWM", "FB", "BAC", "GOOGL"]
    }
]
```

Define Custom Types

To define a custom data type, inherit the `PythonData` class and override the GetSource and Reader methods.

If you need to create multiple objects in your `Reader` method from a single `line` , follow these steps:

1. In the `GetSource` method, pass `FileFormat.UnfoldingCollection` as the third argument to the `SubscriptionDataSource` constructor.
2. In the `Reader` method, order the objects by their timestamp and then return a `BaseDataCollection(endTime, config.Symbol, objects)` where `objects` is a list of your custom data objects.

```
class StockDataSource(PythonData):

    def GetSource(self,
         config: SubscriptionDataConfig,
         date: datetime,
         isLive: bool) -> SubscriptionDataSource:
        return SubscriptionDataSource("https://www.dropbox.com/s/7xe7lfac52mdfpe/custom-universe.json?
dl=1",
                                      SubscriptionTransportMedium.RemoteFile,
                                      FileFormat.UnfoldingCollection)

    def Reader(self,
         config: SubscriptionDataConfig,
         line: str,
         date: datetime,
         isLive: bool) -> BaseData:

        objects = []
        data = json.loads(line)

        for datum in data:
            stocks = StockDataSource()
            stocks.Symbol = config.Symbol

            stocks.Time = datetime.strptime(datum["Date"], "%Y%m%d")
            stocks.EndTime = stocks.Time + timedelta(1)
            stocks["Symbols"] = datum["Symbols"]
            objects.append(stocks)

        return BaseDataCollection(objects[-1].EndTime, config.Symbol, objects)
```

Initialize Universe

To perform a universe selection with custom data, in the `Initialize` method, call the `AddUniverse` method.

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.AddUniverse(StockDataSource, "my-stock-data-source", Resolution.Daily, self.FilterFunction)
```

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points into a `List[StockDataSource])` object it passes to your algorithm's filter function. Your filter function needs to return a list of `Symbol` or `str` object. LEAN automatically subscribes to these new assets and adds them to your algorithm.

```
class MyAlgorithm(QCAlgorithm):
    def FilterFunction(self, data: List[StockDataSource]) -> List[str]:
        list = []
        for item in data:
            for symbol in item["Symbols"]:
                list.append(symbol)
        return list
```

If you add custom properties to your data object in the `Reader` method, LEAN adds them as members to the data object in your filter method. To ensure the property names you add in the `Reader` method follow the convention of member names, LEAN applies the following changes to the property names you provide in the `Reader` method:

1. `-` and `.` characters are replaced with whitespace.

2. The first letter is capitalized.
3. Whitespace characters are removed.

For example, if you set a property name in the `Reader` method to `['some-property.name']` , you can access it in your filter method through the `Somepropertyname` member of your data object.

Demonstration Algorithms

DropboxBaseDataUniverseSelectionAlgorithm.py Python

Importing Data

Bulk Downloads

Introduction

There are two techniques to import data into your algorithm. You can either manually import the entire file or stream the file line-by-line into your algorithm's `OnData` event. This page explores importing an entire file for manual use.

Recommended Use Cases

The batch import technique is outside of the LEAN's awareness or control, so it can't enforce good practices. However, the batch import technique is good for the loading the following datasets:

- Trained AI Models
- Well-defined historical price datasets
- Parameters and setting imports such as `Symbol` lists

Download Files

The `Download` method downloads the content served from a local file or URL and then returns it as a string.

Basic Usage

```py
file = self.Download("<filePathOrURL>")

# If your file is in CSV format, convert it to a DataFrame with the `read_csv` method.
from io import StringIO
import pandas as pd
df = pd.read_csv(StringIO(file))

# If your file is in JSON format, parse it with the `loads` method.
import json
data = json.loads(file)

# If your file is in XML format, parse it with the `fromstring` method.
import xml.etree.ElementTree as ET
root = ET.fromstring(file)
```

Download Method Arguments

The `Download` method can accept header settings, a username, and a password for authentication.

Argument	Data Type	Description	Default Value
address	str	A string containing the URI to download	
headers	Dict[str,str]	Defines header values to add to the request	dict()
userName	str	The user name associated with the credentials	None
password	str	The password for the user name associated with the credentials	None

Download Request Headers

```
header = { "1": "1" }
self.Download(address, headers)
self.Download(address, headers, user_name, password)
```

Transport Binary Data

Follow these steps to transport binary files:

1. Add the following imports to your program:

```
import pickle
import base64
```

2. Serialize your object.

```
pickle_bytes = pickle.dumps(my_object)
base64_str = base64.b64encode(pickle_bytes).decode('ascii')
```

3. Save the string representation of your object into the ObjectStore or one of the supported external sources .
4. Load the string representation of your object into your trading algorithm.
5. Restore the object.

```
base64_bytes = base64_str.encode('ascii')
model = base64.b64decode(base64_bytes)
restored_model = pickle.loads(model)
```

Examples

Demonstration Algorithms
DropboxCoarseFineAlgorithm.py Python NLTKSentimentTradingAlgorithm.py Python

Consolidating Data

Consolidating Data

Getting Started

Introduction

Consolidating data allows you to create bars of any length from smaller bars. Consolidation is commonly used to combine one-minute price bars into longer bars such as 10-20 minute bars. Consolidated bars are helpful because price movement over a longer period can sometimes contain less noise and bars of exotic length are less researched than standard bars, so they may present more opportunities to capture alpha.

To consolidate data, create a `Consolidator` object and register it for data. The built-in consolidators make it easy to create consolidated bars without introducing bugs. In the following sections, we will introduce the different types of consolidators and show you how to shape data into any form.

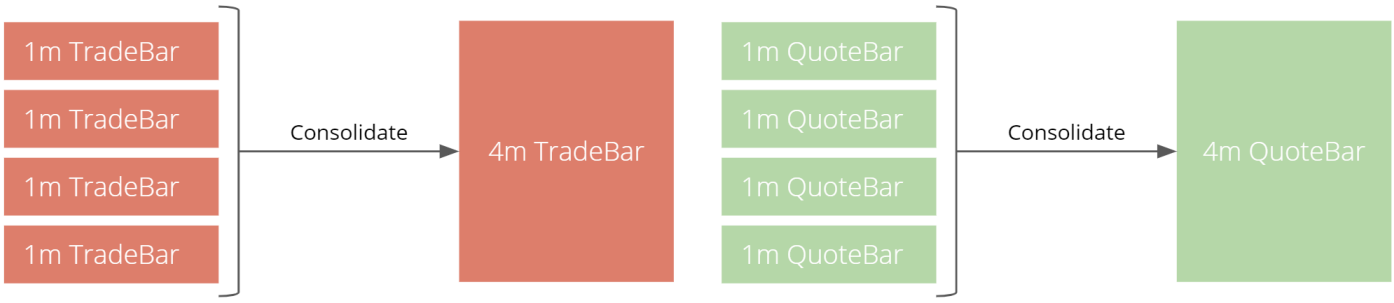
Basic Usage

The following code snippet demonstrates a simple consolidator that aggregates 1-minute bars into 10-minutes bars:

```
# In Initialize, subscribe to a security, create the consolidator, and register it for automatic updates
self.symbol = self.AddEquity("SPY", Resolution.Minute).Symbol
self.consolidator = self.Consolidate(self.symbol, timedelta(minutes=10), self.consolidation_handler)

# Define the consolidation handler
def consolidation_handler(self, consolidated_bar: TradeBar) -> None:
    pass
```

Data Shapes and Sizes

Consolidators usually produce output data that is the same format as the input data. They aggregate small `TradeBar` objects into a large `TradeBar` , small `QuoteBar` objects into a large `QuoteBar` , and `Tick` objects into either a `TradeBar` or `QuoteBar` .



Many asset classes in QuantConnect have data for both trades and quotes. By default, consolidators aggregate data

into `TradeBar` objects. Forex data is the only exception, which consolidators aggregate into `QuoteBar` objects since Forex data doesn't have `TradeBar` objects.

Consolidator Types

There are many different types of consolidators you can create. The process to create and update the consolidator depends on the input data format, output data format, and the consolidation technique.

Time Period Consolidators

Time period consolidators aggregate data based on a period of time like a number of seconds, minutes, or days. If a time period consolidator aggregates data over multiple days, the multi-day aggregation cycle starts and repeats based on the time stamp of the first data point you use to update the consolidator.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Calendar Consolidators

Calendar consolidators aggregate data based on specific periods of time like a weekly basis, quarterly basis, or any schedule with specific start and end times. In contrast to time period consolidators, the bars that calendar consolidators produce always end on the same schedule, regardless of the first data point the consolidator receives.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Count Consolidators

Count consolidators aggregate data based on a number of bars or ticks. This type of consolidator aggregates **n** samples together, regardless of the time period the samples cover. Managing count consolidators is similiar to managing time period consolidators , except you create count consolidators with an integer argument instead of a time-based argument.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Mixed-Mode Consolidators

Mixed-mode consolidators are a combination of count consolidators and time period consolidators. This type of consolidator aggregates **n** samples together or aggregates samples over a specific time period, whichever happens first.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Renko Consolidators

Most Renko consolidators aggregate bars based on a fixed price movement. The `RenkoConsolidator` produces Renko bars by their traditional definition. In the case of a $1 bar size, the `RenkoConsolidator` produces bars that have a body spanning 1.*Theopeningpriceofthefirstbarissettotheclosest*1 multiple of the first trade. When the price moves by at least 1, *thefirstbarcloses*. *Ifabarisarisingbar*, *thefollowingbarcloseswhenthepricemoves*1 above the closing price of the previous bar or 1*belowtheopeningpriceofthepreviousbar*. *Ifabarisafallingbar*, *thefollowingbarcloseswhenthepricemoves*1 below the closing price of the previous bar or $1 above the opening price of the previous bar. If the price jumps multiple dollars in a single tick, the `RenkoConsolidator` produces multiple $1 bars in a single time step.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	RenkoConsolidator
`QuoteBar`	`QuoteBar`	RenkoConsolidator
`Tick`	`TradeBar`	RenkoConsolidator
`Tick`	`QuoteBar`	RenkoConsolidator

Classic Renko Consolidators

Most Renko consolidators aggregate bars based on a fixed price movement. The `ClassicRenkoConsolidator` produces a different type of Renko bars than the `RenkoConsolidator` . A `ClassicRenkoConsolidator` with a bar size of 1*producesanewbarthatspans*1 every time an asset closes $1 away from the close of the previous bar. If the price jumps multiple dollars in a single tick, the `ClassicRenkoConsolidator` only produces one bar per time step where the open of each bar matches the close of the previous bar.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	ClassicRenkoConsolidator
`QuoteBar`	`QuoteBar`	ClassicRenkoConsolidator
`Tick`	`TradeBar`	ClassicRenkoConsolidator
`Tick`	`QuoteBar`	ClassicRenkoConsolidator

Volume Renko Consolidators

Volume Renko consolidators aggregate bars based on a fixed trading volume. These types of bars are commonly known as volume bars. A `VolumeRenkoConsolidator` with a bar size of 10,000 produces a new bar every time 10,000 units of the security trades in the market. If the trading volume of a single time step exceeds two step sizes (i.e. >20,000), the `VolumeRenkoConsolidator` produces multiple bars in a single time step.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	VolumeRenkoConsolidator
`Tick`	`TradeBar`	VolumeRenkoConsolidator

Sequential Consolidators

Sequential consolidators wire two internal consolidators together such that the output of the first consolidator is the input to the second consolidator and the output of the second consolidator is the output of the sequential consolidator.

For more information about sequential consolidators, see Combining Consolidators .

Execution Sequence

The algorithm manager calls events in the following order:

1. Scheduled Events
2. Consolidation event handlers
3. `OnData` event handler

This event flow is important to note. For instance, if your consolidation handlers or `OnData` event handler appends data

to a `RollingWindow` and you use that `RollingWindow` in your Scheduled Event, when the Scheduled Event executes, the `RollingWindow` won't contain the most recent data.

The consolidators are called in the order that they are updated. If you register them for automatic updates, they are updated in the order that you register them.

Examples

Demonstration Algorithms
BasicTemplateFuturesConsolidationAlgorithm.py Python BasicTemplateOptionsConsolidationAlgorithm.py Python
DataConsolidationAlgorithm.py Python MultipleSymbolConsolidationAlgorithm.py Python
Video Tutorials
Opening Range Breakout - Time Period Consolidation Python

Consolidating Data

Consolidator Types

Consolidator Types

Time Period Consolidators

Introduction

Time period consolidators aggregate data based on a period of time like a number of seconds, minutes, or days. If a time period consolidator aggregates data over multiple days, the multi-day aggregation cycle starts and repeats based on the time stamp of the first data point you use to update the consolidator.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `timedelta` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

- `timedelta` Periods

```
self.consolidator = TradeBarConsolidator(timedelta(days=1))
# Aliases:
# self.consolidator = self.CreateConsolidator(timedelta(days=1), TradeBar)
# self.consolidator = self.ResolveConsolidator(self.symbol, timedelta(days=1))
```

  The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

- `Resolution` Periods

  The `Resolution` enumeration has the following members:

```
self.consolidator = TradeBarConsolidator.FromResolution(Resolution.Daily)
# Alias:
# self.consolidator = self.ResolveConsolidator(self.symbol, Resolution.Daily)
```

If the security subscription in your algorithm provides `TradeBar` and `QuoteBar` data, `ResolveConsolidator` returns a `TradeBarConsolidator` .

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at 9:31 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until 9:31 AM ET.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    trade_bar = slice.Bars[self.symbol]
    self.consolidator.Update(trade_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
for trade_bar in history:
    self.consolidator.Update(trade_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `timedelta` or `Resolution` object:

- `timedelta` Periods

```
self.consolidator = self.Consolidate(self.symbol, timedelta(days=1), self.consolidation_handler)
```

- `Resolution` Periods

```
self.consolidator = self.Consolidate(self.symbol, Resolution.Daily, self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: TradeBar) -> None:
    pass
```

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `timedelta` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

- `timedelta` Periods

```
self.consolidator = QuoteBarConsolidator(timedelta(days=1))
# Aliases:
# self.consolidator = self.CreateConsolidator(timedelta(days=1), QuoteBar)
# self.consolidator = self.ResolveConsolidator(self.symbol, timedelta(days=1))
```

  The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

- `Resolution` Periods

  The `Resolution` enumeration has the following members:

```
self.consolidator = self.ResolveConsolidator(self.symbol, Resolution.Daily)
```

If the security subscription in your algorithm provides `TradeBar` and `QuoteBar` data, `ResolveConsolidator` returns a `TradeBarConsolidator` .

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at 9:31 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until 9:31 AM ET.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```
  PY

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      quote_bar = slice.QuoteBars[self.symbol]
      self.consolidator.Update(quote_bar)

  # Example 2: Update the consolidator with data from a history request
  history = self.History[QuoteBar](self.symbol, 30, Resolution.Minute)
  for quote_bar in history:
      self.consolidator.Update(quote_bar)
  ```
  PY

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```
PY

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `timedelta` or `Resolution` object:

- `timedelta` Periods

  ```
  self.consolidator = self.Consolidate(self.symbol, timedelta(days=1), TickType.Quote,
  self.consolidation_handler)
  ```
  PY

- `Resolution` Periods

```
self.consolidator = self.Consolidate(self.symbol, Resolution.Daily, TickType.Quote,
self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: QuoteBar) -> None:
    pass
```

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `timedelta` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

- `timedelta` Periods

  ```
  self.consolidator = TickConsolidator(timedelta(milliseconds=100))
  # Aliases:
  # self.consolidator = self.CreateConsolidator(timedelta(milliseconds=100), Tick)
  # self.consolidator = self.ResolveConsolidator(self.symbol, timedelta(milliseconds=100))
  ```

  The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

- `Resolution` Periods

  The `Resolution` enumeration has the following members:

  ```
  self.consolidator = self.ResolveConsolidator(self.symbol, Resolution.Second)
  ```

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the

consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
PY
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars milliseconds after 9:30 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until milliseconds after 9:30 AM ET.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  PY
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  PY
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      ticks = slice.Ticks[self.symbol]
      for tick in ticks:
       self.consolidator.Update(tick)

  # Example 2: Update the consolidator with data from a history request
  ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
  for tick in ticks:
      self.consolidator.Update(tick)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `timedelta` or `Resolution` object:

- `timedelta` Periods

  ```
  self.consolidator = self.Consolidate(self.symbol, timedelta(milliseconds=100),
  self.consolidation_handler)
  ```

- `Resolution` Periods

  ```
  self.consolidator = self.Consolidate(self.symbol, Resolution.Second, self.consolidation_handler)
  ```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: TradeBar) -> None:
    pass
```

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `timedelta` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   - `timedelta` Periods

     ```
     self.consolidator = TickQuoteBarConsolidator(timedelta(milliseconds=100))
     # Aliases:
     # self.consolidator = self.CreateConsolidator(timedelta(milliseconds=100), Tick, TickType.Quote)
     # self.consolidator = self.ResolveConsolidator(self.symbol, timedelta(milliseconds=100))
     ```

     The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

- `Resolution` Periods

  The `Resolution` enumeration has the following members:

  ```
  self.consolidator = self.ResolveConsolidator(self.symbol, Resolution.Second)
  ```

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars milliseconds after 9:30 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until milliseconds after 9:30 AM ET.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `timedelta` or `Resolution` object:

- `timedelta` Periods

```
self.consolidator = self.Consolidate(self.symbol, timedelta(milliseconds=100), TickType.Quote,
self.consolidation_handler)
```

- `Resolution` Periods

```
self.consolidator = self.Consolidate(self.symbol, Resolution.Second, TickType.Quote,
self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: QuoteBar) -> None:
    pass
```

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow

these steps to create and manage a <dataType> consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a timedelta or Resolution object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

- timedelta Periods

```
PY
self.consolidator = DynamicDataConsolidator(timedelta(days=1))
# Aliases:
# self.consolidator = self.CreateConsolidator(timedelta(days=1), <dataType>)
# self.consolidator = self.ResolveConsolidator(self.symbol, timedelta(days=1))
```

  The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

- Resolution Periods

  The Resolution enumeration has the following members:

```
PY
self.consolidator = self.ResolveConsolidator(self.symbol, Resolution.Daily)
```

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

- Add an event handler to the consolidator.

```
PY
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis () at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be self.consolidation_handler , not self.consolidation_handler() .

- Define the consolidation handler.

```
PY
def consolidation_handler(self, sender: object, consolidated_bar: <dataType>) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone .

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  PY
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  PY
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      if self.symbol in slice:
          self.consolidator.Update(slice[self.symbol])

  # Example 2: Update the consolidator with data from a history request
  history = self.History[<dataType>](self.symbol, 30, Resolution.Daily)
  for data in history:
      self.consolidator.Update(data)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
PY
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `timedelta` or `Resolution` object:

- `timedelta` Periods

  ```
  PY
  self.consolidator = self.Consolidate(self.symbol, timedelta(days=1), self.consolidation_handler)
  ```

- `Resolution` Periods

  ```
  PY
  self.consolidator = self.Consolidate(self.symbol, Resolution.Daily, self.consolidation_handler)
  ```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: <dataType>) -> None:
    pass
```

PY

Examples

Demonstration Algorithms
BasicTemplateOptionsConsolidationAlgorithm.py Python DataConsolidationAlgorithm.py Python
MultipleSymbolConsolidationAlgorithm.py Python

Consolidator Types

Calendar Consolidators

Introduction

Calendar consolidators aggregate data based on specific periods of time like a weekly basis, quarterly basis, or any schedule with specific start and end times. In contrast to time period consolidators, the bars that calendar consolidators produce always end on the same schedule, regardless of the first data point the consolidator receives.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

- Standard Periods

  The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
self.consolidator = TradeBarConsolidator(Calendar.Weekly)
# Alias:
# self.consolidator = self.CreateConsolidator(Calendar.Weekly, TradeBar)
```

PY

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `datetime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the

duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
# Define a consolidation period method
def consolidation_period(self, dt: datetime) -> CalendarInfo:
    period = timedelta(7)

    dt = dt.replace(hour=17, minute=0, second=0, microsecond=0)
    delta = 1+dt.weekday()
    if delta > 6:
        delta = 0
    start = dt-timedelta(delta)

    return CalendarInfo(start, period)

# Create the consolidator with the consolidation period method
self.consolidator = TradeBarConsolidator(self.consolidation_period)
```

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    pass
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
PY
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    trade_bar = slice.Bars[self.symbol]
    self.consolidator.Update(trade_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
for trade_bar in history:
    self.consolidator.Update(trade_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
PY
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
PY
self.consolidator = self.Consolidate(self.symbol, Calendar.Weekly, self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
PY
def consolidation_handler(self, consolidated_bar: TradeBar) -> None:
    pass
```

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

- Standard Periods

  The following table describes the helper methods that the `Calendar` class provides to create the built-in

`CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
PY
self.consolidator = QuoteBarConsolidator(Calendar.Weekly)
# Alias:
# self.consolidator = self.CreateConsolidator(Calendar.Weekly, QuoteBar)
```

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `datetime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
PY
# Define a consolidation period method
def consolidation_period(self, dt: datetime) -> CalendarInfo:
    period = timedelta(7)

    dt = dt.replace(hour=17, minute=0, second=0, microsecond=0)
    delta = 1+dt.weekday()
    if delta > 6:
        delta = 0
    start = dt-timedelta(delta)

    return CalendarInfo(start, period)

# Create the consolidator with the consolidation period method
self.consolidator = QuoteBarConsolidator(self.consolidation_period)
```

- Add an event handler to the consolidator.

```
PY
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be

`self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
PY
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  PY
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  PY
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      quote_bar = slice.QuoteBars[self.symbol]
      self.consolidator.Update(quote_bar)

  # Example 2: Update the consolidator with data from a history request
  history = self.History[QuoteBar](self.symbol, 30, Resolution.Minute)
  for quote_bar in history:
      self.consolidator.Update(quote_bar)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
PY
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
self.consolidator = self.Consolidate(self.symbol, Calendar.Weekly, TickType.Quote,
self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: QuoteBar) -> None:
    pass
```

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

- Standard Periods

  The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
self.consolidator = TickConsolidator(Calendar.Weekly)
# Alias:
# self.consolidator = self.CreateConsolidator(Calendar.Weekly, Tick)
```

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `datetime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the

duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
# Define a consolidation period method
def consolidation_period(self, dt: datetime) -> CalendarInfo:
    period = timedelta(7)

    dt = dt.replace(hour=17, minute=0, second=0, microsecond=0)
    delta = 1+dt.weekday()
    if delta > 6:
        delta = 0
    start = dt-timedelta(delta)

    return CalendarInfo(start, period)

# Create the consolidator with the consolidation period method
self.consolidator = TickConsolidator(self.consolidation_period)
```

PY

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

PY

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    pass
```

PY

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

  PY

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
self.consolidator = self.Consolidate(self.symbol, Calendar.Weekly, self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: TradeBar) -> None:
    pass
```

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

- Standard Periods

  The following table describes the helper methods that the `Calendar` class provides to create the built-in

`CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
PY
self.consolidator = TickQuoteBarConsolidator(Calendar.Weekly)
# Alias:
# self.consolidator = self.CreateConsolidator(Calendar.Weekly, Tick, TickType.Quote)
```

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `datetime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
PY
# Define a consolidation period method
def consolidation_period(self, dt: datetime) -> CalendarInfo:
    period = timedelta(7)

    dt = dt.replace(hour=17, minute=0, second=0, microsecond=0)
    delta = 1+dt.weekday()
    if delta > 6:
        delta = 0
    start = dt-timedelta(delta)

    return CalendarInfo(start, period)

# Create the consolidator with the consolidation period method
self.consolidator = TickQuoteBarConsolidator(self.consolidation_period)
```

- Add an event handler to the consolidator.

```
PY
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be

`self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      ticks = slice.Ticks[self.symbol]
      for tick in ticks:
       self.consolidator.Update(tick)

  # Example 2: Update the consolidator with data from a history request
  ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
  for tick in ticks:
      self.consolidator.Update(tick)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
self.consolidator = self.Consolidate(self.symbol, Calendar.Weekly, TickType.Quote,
self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: QuoteBar) -> None:
    pass
```

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a `<dataType>` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

- Standard Periods

  The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
self.consolidator = DynamicDataConsolidator(Calendar.Weekly)
# Alias:
# self.consolidator = self.CreateConsolidator(Calendar.Weekly, <dataType>)
```

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `datetime` object that's based in the data time zone

and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
PY
# Define a consolidation period method
def consolidation_period(self, dt: datetime) -> CalendarInfo:
    period = timedelta(7)

    dt = dt.replace(hour=17, minute=0, second=0, microsecond=0)
    delta = 1+dt.weekday()
    if delta > 6:
        delta = 0
    start = dt-timedelta(delta)

    return CalendarInfo(start, period)

# Create the consolidator with the consolidation period method
self.consolidator = DynamicDataConsolidator(self.consolidation_period)
```

- Add an event handler to the consolidator.

```
PY
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
PY
def consolidation_handler(self, sender: object, consolidated_bar: <dataType>) -> None:
    pass
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  PY
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      if self.symbol in slice:
          self.consolidator.Update(slice[self.symbol])

  # Example 2: Update the consolidator with data from a history request
  history = self.History[<dataType>](self.symbol, 30, Resolution.Daily)
  for data in history:
      self.consolidator.Update(data)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
self.consolidator = self.Consolidate(self.symbol, Calendar.Weekly, self.consolidation_handler)
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
def consolidation_handler(self, consolidated_bar: <dataType>) -> None:
    pass
```

Examples

The following examples are typical calendar functions you may want.

The function should receive a `datetime` object that's based in the algorithm time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. This function is evaluated when the duration of the consolidation period has passed and at the following time step.

The preceding sections of this page provide a typical calendar function that consolidates data weekly, starting at 5:00 PM. If you consolidate US Equity and Crypto data, the event handler triggers at different times since the data time zone

of US Equity is America/New York and Crypto is UTC.

Example 1: Consolidate data into 20-minute `TradeBar` objects that start at 9:30 AM

This is a typical case for the US markets because they open every business day at 9:30 PM Eastern Time (ET).

```py
def every_twenty_minute_after_equity_market_open(self, dt: datetime) -> CalendarInfo:
    period = timedelta(minutes=20)
    open_time = dt.replace(hour=9, minute=30, second=0, microsecond=0)
    start = open_time + int((dt - open_time) / period) * period
    return CalendarInfo(start, period)
```

To consolidate data into another period, change the `period` variable. For example, to consolidate into 4-hour bars, replace `timedelta(minutes=20)` with `timedelta(hours=4)`

Example 2: Consolidate data in daily `QuoteBar` objects that start at 5 PM:

This is a typical Forex case because the Forex market opens on Sunday at 5 PM ET.

```py
def custom_daily_forex(self, dt: datetime) -> CalendarInfo:
start = dt.replace(hour=17, minute=0, second=0)
if dt.hour < 17:
    start -= timedelta(1)
return CalendarInfo(start, timedelta(1))
```

Example 3: Consolidate data into monthly `TradeBar` objects that start at midnight:

```py
def custom_monthly(self, dt: datetime) -> CalendarInfo:
    start = dt.replace(day=1).date()
    end = dt.replace(day=28) + timedelta(4)
    end = (end - timedelta(end.day - 1)).date()
    return CalendarInfo(start, end - start)
```

Demonstration Algorithms
DataConsolidationAlgorithm.py Python

Consolidator Types

Count Consolidators

Introduction

Count consolidators aggregate data based on a number of bars or ticks. This type of consolidator aggregates **n** samples together, regardless of the time period the samples cover. Managing count consolidators is similiar to managing time period consolidators , except you create count consolidators with an integer argument instead of a time-based argument.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   self.consolidator = TradeBarConsolidator(10)
   ```

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

   ```
   def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
       pass
   ```

   When the consolidator receives the **n** -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    trade_bar = slice.Bars[self.symbol]
    self.consolidator.Update(trade_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
for trade_bar in history:
    self.consolidator.Update(trade_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

```
self.consolidator = QuoteBarConsolidator(10)
```

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
PY
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

When the consolidator receives the **n** -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
PY
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
PY
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    quote_bar = slice.QuoteBars[self.symbol]
    self.consolidator.Update(quote_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[QuoteBar](self.symbol, 30, Resolution.Minute)
for quote_bar in history:
    self.consolidator.Update(quote_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
PY
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to

slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   self.consolidator = TickConsolidator(10)
   ```
   PY

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```
   PY

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

   ```
   def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
       pass
   ```
   PY

   When the consolidator receives the **n** -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```
  PY

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To

manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   self.consolidator = TickQuoteBarConsolidator(10)
   ```

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

When the consolidator receives the **n** -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a `<dataType>` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   self.consolidator = DynamicDataConsolidator(10)
   ```
   PY

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```
   PY

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

   ```
   def consolidation_handler(self, sender: object, consolidated_bar: <dataType>) -> None:
       pass
   ```
   PY

   When the consolidator receives the **n** -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```
  PY

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice:
        self.consolidator.Update(slice[self.symbol])

# Example 2: Update the consolidator with data from a history request
history = self.History[<dataType>](self.symbol, 30, Resolution.Daily)
for data in history:
    self.consolidator.Update(data)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Examples

Demonstration Algorithms
DataConsolidationAlgorithm.py Python

Consolidator Types

Mixed-Mode Consolidators

Introduction

Mixed-mode consolidators are a combination of count consolidators and time period consolidators. This type of consolidator aggregates **n** samples together or aggregates samples over a specific time period, whichever happens first.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `timedelta` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   self.consolidator = TradeBarConsolidator(10, timedelta(days=1))
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at 9:31 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until 9:31 AM ET.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    trade_bar = slice.Bars[self.symbol]
    self.consolidator.Update(trade_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
for trade_bar in history:
    self.consolidator.Update(trade_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe

subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `timedelta` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   self.consolidator = QuoteBarConsolidator(10, timedelta(days=1))
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

   ```
   def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
       pass
   ```

   The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at 9:31 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your

algorithm doesn't receive minute resolution data after 4:00 PM ET until 9:31 AM ET.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    quote_bar = slice.QuoteBars[self.symbol]
    self.consolidator.Update(quote_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[QuoteBar](self.symbol, 30, Resolution.Minute)
for quote_bar in history:
    self.consolidator.Update(quote_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `timedelta` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute

bars.

```
self.consolidator = TickConsolidator(10, timedelta(milliseconds=100))
```

The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars milliseconds after 9:30 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until milliseconds after 9:30 AM ET.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `timedelta` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

```
self.consolidator = TickQuoteBarConsolidator(10, timedelta(milliseconds=100))
```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time

(UTC), regardless of the algorithm time zone.

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: QuoteBar) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars milliseconds after 9:30 AM Eastern Time (ET). The consolidated bar for US Equities doesn't close at 4:00 PM ET because the day isn't over. The consolidated bar for US Equities also doesn't close at midnight because your algorithm doesn't receive minute resolution data after 4:00 PM ET until milliseconds after 9:30 AM ET.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with

data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a `<dataType>` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `timedelta` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   self.consolidator = DynamicDataConsolidator(10, timedelta(days=1))
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET.

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: <dataType>) -> None:
    pass
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice:
        self.consolidator.Update(slice[self.symbol])

# Example 2: Update the consolidator with data from a history request
history = self.History[<dataType>](self.symbol, 30, Resolution.Daily)
for data in history:
    self.consolidator.Update(data)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidator Types

Renko Consolidators

Renko Consolidators

Renko Consolidators

Introduction

Most Renko consolidators aggregate bars based on a fixed price movement. The `RenkoConsolidator` produces Renko bars by their traditional definition. In the case of a $1 bar size, the `RenkoConsolidator` produces bars that have a body spanning 1.*Theopeningpriceofthefirstbarissettotheclosest*1 multiple of the first trade. When the price moves by at least 1, *thefirstbarcloses*. *Ifabarisarisingbar*, *thefollowingbarcloseswhenthepricemoves*1 above the closing price of the previous bar or 1*belowtheopeningpriceofthepreviousbar*. *Ifabarisafallingbar*, *thefollowingbarcloseswhenthepricemoves*1 below the closing price of the previous bar or $1 above the opening price of the previous bar. If the price jumps multiple dollars in a single tick, the `RenkoConsolidator` produces multiple $1 bars in a single time step.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

   ```
   # Create a Renko consolidator that emits a bar when the price moves $1
   self.consolidator = RenkoConsolidator(1)
   ```

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    trade_bar = slice.Bars[self.symbol]
    self.consolidator.Update(trade_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
for trade_bar in history:
    self.consolidator.Update(trade_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `QuoteBar` consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

```
# Create a Renko consolidator that emits a bar when the price moves $1
self.consolidator = RenkoConsolidator(1)
```

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    quote_bar = slice.QuoteBars[self.symbol]
    self.consolidator.Update(quote_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[QuoteBar](self.symbol, 30, Resolution.Minute)
for quote_bar in history:
    self.consolidator.Update(quote_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `RenkoBar` objects. Follow these steps to create and manage a `Tick` consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

   ```
   # Create a Renko consolidator that emits a bar when the price moves $1
   self.consolidator = RenkoConsolidator(1)
   ```

2. Add an event handler to the consolidator.

   ```
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

   ```
   def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
       pass
   ```

The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `RenkoBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

```
# Create a Renko consolidator that emits a bar when the price moves $1
self.consolidator = RenkoConsolidator(1)
```

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Renko Consolidators

Classic Renko Consolidators

Introduction

Most Renko consolidators aggregate bars based on a fixed price movement. The `ClassicRenkoConsolidator` produces a different type of Renko bars than the `RenkoConsolidator` . A `ClassicRenkoConsolidator` with a bar size of 1*producesanewbarthatspans*1 every time an asset closes $1 away from the close of the previous bar. If the price jumps multiple dollars in a single tick, the `ClassicRenkoConsolidator` only produces one bar per time step where the open of each bar matches the close of the previous bar.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   # Create a Classic Renko consolidator that emits a bar when the price moves $1
   self.consolidator = ClassicRenkoConsolidator(1)
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

- It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
- It ignores the volume of the input data
- It enforces the open and close of each bar to be a multiple of the bar size

To build the Renko bars with a different property than the `Value` of the `IBaseData` object, provide a `selector` argument. The `selector` should be a function that receives the `IBaseData` object and returns a decimal value.

```
self.consolidator = ClassicRenkoConsolidator(1, selector = lambda data: data.High)
```

To add a non-zero `Volume` property to the Renko bars, provide a `volumeSelector` argument. The `volumeSelector` should be a function that receives the `IBaseData` object and returns a decimal value.

```
self.consolidator = ClassicRenkoConsolidator(1, volumeSelector = lambda data: data.Volume)
```

To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` .

The following opening and closing Renko bar values are all multiples of the first value from the `selector`

```
self.consolidator = ClassicRenkoConsolidator(1, evenBars = False)
```

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    trade_bar = slice.Bars[self.symbol]
    self.consolidator.Update(trade_bar)

# Example 2: Update the consolidator with data from a history request
history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
for trade_bar in history:
    self.consolidator.Update(trade_bar)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `QuoteBar` consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   # Create a Classic Renko consolidator that emits a bar when the price moves $1
   self.consolidator = ClassicRenkoConsolidator(1)
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

- It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
- It ignores the volume of the input data
- It enforces the open and close of each bar to be a multiple of the bar size

The following arguments enable you to create Renko bars that aggregate the excess liquidity on the bid.

```
self.consolidator = ClassicRenkoConsolidator(10, lambda data: data.Value, lambda data: data.LastBidSize - data.LastAskSize)
```

To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

```
self.consolidator = ClassicRenkoConsolidator(1, evenBars = False)
```

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      quote_bar = slice.QuoteBars[self.symbol]
      self.consolidator.Update(quote_bar)

  # Example 2: Update the consolidator with data from a history request
  history = self.History[QuoteBar](self.symbol, 30, Resolution.Minute)
  for quote_bar in history:
      self.consolidator.Update(quote_bar)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `RenkoBar` objects. Follow these steps to create and manage a `Tick` consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   # Create a Classic Renko consolidator that emits a bar when the price moves $1
   self.consolidator = ClassicRenkoConsolidator(1)
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

- It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
- It ignores the volume of the input data
- It enforces the open and close of each bar to be a multiple of the bar size

To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

```
self.consolidator = ClassicRenkoConsolidator(1, evenBars = False)
```

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      ticks = slice.Ticks[self.symbol]
      for tick in ticks:
       self.consolidator.Update(tick)

  # Example 2: Update the consolidator with data from a history request
  ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
  for tick in ticks:
      self.consolidator.Update(tick)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `RenkoBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   # Create a Classic Renko consolidator that emits a bar when the price moves $1
   self.consolidator = ClassicRenkoConsolidator(1)
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

- It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
- It ignores the volume of the input data
- It enforces the open and close of each bar to be a multiple of the bar size

To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

```
self.consolidator = ClassicRenkoConsolidator(1, evenBars = False)
```

- Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

- Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: RenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

- Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      ticks = slice.Ticks[self.symbol]
      for tick in ticks:
       self.consolidator.Update(tick)

  # Example 2: Update the consolidator with data from a history request
  ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
  for tick in ticks:
      self.consolidator.Update(tick)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Examples

Demonstration Algorithms
ClassicRenkoConsolidatorAlgorithm.py Python

Renko Consolidators

Volume Renko Consolidators

Introduction

Volume Renko consolidators aggregate bars based on a fixed trading volume. These types of bars are commonly known as volume bars. A `VolumeRenkoConsolidator` with a bar size of 10,000 produces a new bar every time 10,000 units of the security trades in the market. If the trading volume of a single time step exceeds two step sizes (i.e. >20,000), the `VolumeRenkoConsolidator` produces multiple bars in a single time step.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `VolumeRenkoBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on custom set volume traded:

1. Create the consolidator.

   To create a Volume Renko consolidator, pass the bar size to the `VolumeRenkoConsolidator` constructor.

   ```
   PY
   # Create a Volume Renko consolidator that emits a bar every time 10,000 units trade
   self.consolidator = VolumeRenkoConsolidator(10000)
   ```

2. Add an event handler to the consolidator.

   ```
   PY
   self.consolidator.DataConsolidated += self.consolidation_handler
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

   ```
   PY
   def consolidation_handler(self, sender: object, consolidated_bar: VolumeRenkoBar) -> None:
       pass
   ```

   The consolidation event handler receives bars when the trading volume forms a new Volume Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  PY
  self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  PY
  # Example 1: Update the consolidator with data from the Slice object
  def OnData(self, slice: Slice) -> None:
      trade_bar = slice.Bars[self.symbol]
      self.consolidator.Update(trade_bar)

  # Example 2: Update the consolidator with data from a history request
  history = self.History[TradeBar](self.symbol, 30, Resolution.Minute)
  for trade_bar in history:
      self.consolidator.Update(trade_bar)
  ```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
PY
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `VolumeRenkoBar` objects. Follow these steps to create and manage a `Tick` consolidator based on custom set volume traded:

1. Create the consolidator.

   To create a Volume Renko consolidator, pass the bar size to the `VolumeRenkoConsolidator` constructor.

   ```
   PY
   # Create a Volume Renko consolidator that emits a bar every time 10,000 units trade
   self.consolidator = VolumeRenkoConsolidator(10000)
   ```

2. Add an event handler to the consolidator.

```
self.consolidator.DataConsolidated += self.consolidation_handler
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `self.consolidation_handler` , not `self.consolidation_handler()` .

3. Define the consolidation handler.

```
def consolidation_handler(self, sender: object, consolidated_bar: VolumeRenkoBar) -> None:
    pass
```

The consolidation event handler receives bars when the trading volume forms a new Volume Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
# Example 1: Update the consolidator with data from the Slice object
def OnData(self, slice: Slice) -> None:
    ticks = slice.Ticks[self.symbol]
    for tick in ticks:
     self.consolidator.Update(tick)

# Example 2: Update the consolidator with data from a history request
ticks = self.History[Tick](self.symbol, timedelta(minutes=3), Resolution.Tick)
for tick in ticks:
    self.consolidator.Update(tick)
```

- If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
self.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Examples

Demonstration Algorithms
VolumeRenkoConsolidatorAlgorithm.py Python

Consolidator Types

Combining Consolidators

Introduction

Sequential consolidators wire two internal consolidators together such that the output of the first consolidator is the input to the second consolidator and the output of the second consolidator is the output of the sequential consolidator.

Create Consolidators

To create a sequential consolidator, create two consolidators and then pass them to the `SequentialConsolidator` constructor.

```
# This first consolidator produces a consolidated bar after a day passes
one_day_consolidator = TradeBarConsolidator(timedelta(days=1))

# This second consolidators produces a consolidated bar after it sees 3 samples
three_count_consolidator = TradeBarConsolidator(3)

# This sequential consolidator aggregates three 1-day bars together
self.consolidator = SequentialConsolidator(one_day_consolidator, three_count_consolidator)
```

PY

For more information about each type of consolidator, see Consolidator Types .

Examples

Demonstration Algorithms
DataConsolidationAlgorithm.py Python

Consolidating Data

Consolidator History

Introduction

This page explains how to save and access historical consolidated bars.

Save Consolidated Bars

To access historical bars that were passed to your consolidation handler, save the bars as you receive them. You can use a RollingWindow to save the consolidated bars and easily access them later on in your algorithm.

```
PY

# Create a class member to store the RollingWindow
self.window = RollingWindow[TradeBar](2)

# In the consolidation handler, add consolidated bars to the RollingWindow
def consolidation_handler(self, sender: object, consolidated_bar: TradeBar) -> None:
    self.window.Add(consolidated_bar)
```

Get Historical Bars

If you save consolidated bars in a `RollingWindow` , you can access them by indexing the `RollingWindow` . `RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window.

```
PY

most_recent_bar = self.window[0]
previous_bar = self.window[1]
oldest_bar = self.window[self.window.Count-1]
```

To get the consolidated bar that was most recently removed from the `RollingWindow` , use the `MostRecentlyRemoved` property.

```
PY

removed_bar = self.window.MostRecentlyRemoved
```

Consolidating Data

Updating Indicators

Introduction

You can use consolidators to automatically update indicators in your algorithms. The consolidators can update your indicators at each time step or with aggregated bars. By default, LEAN updates data point indicators with the close price of the consolidated bars, but you can change it to a custom data field.

Standard Indicator Periods

If your algorithm has a static universe, you can create automatic indicators in just one line of code. When you create an automatic indicator, LEAN creates a consolidator, hooks it up for automatic updates, and then updates the indicator with the consolidated bars.

```
PY
 # Consolidate minute SPY data into 14-bar daily indicators
ema = self.EMA("SPY", 14, Resolution.Daily)
sma = self.SMA("SPY", 14, Resolution.Daily)
```

If your algorithm has a dynamic universe, create a manual indicator and then create a time period consolidator that updates the indicator at each time step. Keep a reference to the consolidator so you can remove it when your algorithm removes the security from the universe.

```
PY
self.ema = ExponentialMovingAverage(14)
self.consolidator = TradeBarConsolidator(1)
self.consolidator.DataConsolidated += lambda _, bar: self.ema.Update(bar.EndTime, bar.Close)
self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)
```

Custom Indicator Periods

It's common to update indicators with price data that spans a normal time period like one minute or one day. It's less common to update an indicator with exotic time periods (for example, a 7-minute consolidated bar), so these types of indicators may provide more opportunities for alpha. To update indicators with exotic data, create a manual indicator and then call the `RegisterIndicator` method. The `RegisterIndicator` method wires up the indicator for automatic updates at the time interval you provide.

```
PY
# Calculate the SMA with 10 7-minute bars
self.symbol = self.AddEquity("SPY", Resolution.Minute).Symbol
self.indicator = SimpleMovingAverage(10)
self.RegisterIndicator(self.symbol, self.indicator, timedelta(minutes=7))
```

The `RegisterIndicator` method can accept a `timedelta` , `Resolution` , or an unregistered consolidator. If you apply the indicator to a security in a dynamic universe, provide a consolidator so that you can remove it when your algorithm

removes the security from the universe.

```py
# timedelta
self.RegisterIndicator(self.symbol, self.indicator, timedelta(minutes=7))

# Resolution
self.RegisterIndicator(self.symbol, self.indicator, Resolution.Hour)

# Consolidator
self.consolidator = TradeBarConsolidator(35)
self.RegisterIndicator(self.symbol, self.indicator, self.consolidator)
```

Custom Indicator Values

Data point indicators use only a single price data in their calculations. By default, those indicators use the closing price. For assets with `TradeBar` data, that price is the `TradeBar` close price. For assets with `QuoteBar` data, that price is the mid-price of the bid closing price and the ask closing price. To create an indicator with the other fields like the `Open` , `High` , `Low` , or `Close` , provide a `selector` argument to the `RegisterIndicator` method.

```py
# Define a 10-period RSI with indicator constructor
self.rsi = RelativeStrengthIndex(10, MovingAverageType.Simple)

# Register the daily High price data to automatically update the indicator
self.RegisterIndicator(self.symbol, self.rsi, Resolution.Daily, Field.High)
```

The `RegisterIndicator` method can accept a `timedelta` , `Resolution` , or an unregistered consolidator. If you apply the indicator to a security in a dynamic universe, provide a consolidator so that you can remove it when your algorithm removes the security from the universe.

The `Field` class has the following `selector` properties:

Historical Data

Historical Data

History Requests

Introduction

There are two ways to request historical data in your algorithms: direct historical data requests and indirect algorithm warm up . You can use a direct historical data request at any time throughout your algorithm. It returns all of the data you request as a single object.

Key History Concepts

The historical data API has many different options to give you the greatest flexibility in how to apply it to your algorithm.

Time Period Options

You can request historical data based on a trailing number of bars, a trailing period of time, or a defined period of time. If you request data in a defined period of time, the `datetime` objects you provide are based in the algorithm time zone .

Return Formats

Each asset class supports slightly different data formats. When you make a history request, consider what data returns. Depending on how you request the data, history requests return a specific data type. For example, if you don't provide `Symbol` objects, you get `Slice` objects that contain the entire universe.

The most popular return type is a `DataFrame` . If you request a `DataFrame` , LEAN unpacks the data from `Slice` objects to populate the `DataFrame` . If you intend to use the data in the `DataFrame` to create `TradeBar` or `QuoteBar` objects, request that the history request returns the data type you need. Otherwise, LEAN will waste computational resources populating the `DataFrame` .

Time Index

When your history request returns a `DataFrame` , the timestamps in the `DataFrame` are based on the data time zone . When your history request returns a `TradeBars` , `QuoteBars` , `Ticks` , or `Slice` object, the `Time` properties of these objects are based on the algorithm time zone, but the `EndTime` properties of the individual `TradeBar` , `QuoteBar` , and `Tick` objects are based on the data time zone . The `EndTime` is the end of the sampling period and when the data is actually available. For daily US Equity data, this results in data points appearing on Saturday and skipping Monday.

Request Data

The simplest form of history request is for a known set of `Symbol` objects. This is common for fixed universes of

securities or when you need to prepare new securities added to your algorithm. History requests return slightly different data depending on the overload you call. The data that returns is in ascending order from oldest to newest. This order is necessary to use the data to warm up indicators.

Single Symbol History Requests

To request history for a single asset, pass the asset `Symbol` to the `History` method. The return type of the method call depends on the history request `[Type]` . The following table describes the return type of each request `[Type]` :

Request Type	Return Data Type
No argument	`DataFrame`
`TradeBar`	`List[TradeBars]`
`QuoteBar`	`List[QuoteBars]`
`Tick`	`List[Ticks]`
`alternativeDataClass` (ex: `CBOE`)	`List[ alternativeDataClass ]` (ex: `List[CBOE]`)

Each row of the DataFrame represents the prices at a point in time. Each column of the DataFrame is a property of that price data (for example, open, high, low, and close (OHLC)). If you request a DataFrame object and pass `TradeBar` as the first argument, the DataFrame that returns only contains the OHLC and volume columns. If you request a DataFrame object and pass `QuoteBar` as the first argument, the DataFrame that returns contains the OHLC of the bid and ask and it contains OHLC columns, which are the respective means of the bid and ask OHLC values. If you request a DataFrame and don't pass `TradeBar` or `QuoteBar` as the first arugment, the DataFrame that returns contains columns for all of the data that's available for the given resolution.

PY

```
# EXAMPLE 1: Requesting By Bar Count: 5 bars at the security resolution:
vix_symbol = self.AddData(CBOE, "VIX", Resolution.Daily).Symbol
cboe_data = self.History[CBOE](vix_symbol, 5)

btc_symbol = self.AddCrypto("BTCUSD", Resolution.Minute).Symbol
trade_bars = self.History[TradeBar](btc_symbol, 5)
quote_bars = self.History[QuoteBar](btc_symbol, 5)
trade_bars_df = self.History(TradeBar, btc_symbol, 5)
quote_bars_df = self.History(QuoteBar, btc_symbol, 5)
df = self.History(btc_symbol, 5)   # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-07-09 04:01:00	21596.65	21607.03	21591.50	21602.53	0.229349	21593.66	21604.13	21590.42	21600.24	0.180000	21596.65	21605.56	21591.51	21599.96	2.299479
	2022-07-09 04:02:00	21611.12	21611.13	21596.58	21596.65	0.460583	21611.11	21611.12	21593.66	21593.66	0.000217	21611.12	21611.12	21596.28	21596.58	4.046283
	2022-07-09 04:03:00	21619.75	21624.05	21606.98	21611.12	0.001000	21619.74	21621.81	21605.30	21611.11	0.000368	21619.75	21621.81	21605.61	21611.12	6.201696
	2022-07-09 04:04:00	21612.18	21623.30	21602.54	21619.75	0.021005	21610.67	21620.54	21602.36	21619.74	0.050000	21610.47	21620.54	21602.87	21619.74	8.323808
	2022-07-09 04:05:00	21608.68	21613.22	21602.01	21612.18	0.002000	21608.67	21612.04	21598.37	21610.67	0.050000	21606.92	21613.05	21598.43	21613.05	2.890605

PY

```
# EXAMPLE 2: Requesting By Bar Count: 5 bars with a specific resolution:
trade_bars = self.History[TradeBar](btc_symbol, 5, Resolution.Daily)
quote_bars = self.History[QuoteBar](btc_symbol, 5, Resolution.Minute)
trade_bars_df = self.History(TradeBar, btc_symbol, 5, Resolution.Minute)
quote_bars_df = self.History(QuoteBar, btc_symbol, 5, Resolution.Minute)
df = self.History(btc_symbol, 5, Resolution.Minute)  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-07-09 04:01:00	21596.65	21607.03	21591.50	21602.53	0.229349	21593.66	21604.13	21590.42	21600.24	0.180000	21596.65	21605.56	21591.51	21599.96	2.299479
	2022-07-09 04:02:00	21611.12	21611.13	21596.58	21596.65	0.460583	21611.11	21611.12	21593.66	21593.66	0.000217	21611.12	21611.12	21596.28	21596.58	4.046283
	2022-07-09 04:03:00	21619.75	21624.05	21606.98	21611.12	0.001000	21619.74	21621.81	21605.30	21611.11	0.000368	21619.75	21621.81	21605.61	21611.12	6.201696
	2022-07-09 04:04:00	21612.18	21623.30	21602.54	21619.75	0.021005	21610.67	21620.54	21602.36	21619.74	0.050000	21610.47	21620.54	21602.87	21619.74	8.323808
	2022-07-09 04:05:00	21608.68	21613.22	21602.01	21612.18	0.002000	21608.67	21612.04	21598.37	21610.67	0.050000	21606.92	21613.05	21598.43	21613.05	2.890605

PY

```
# EXAMPLE 3: Requesting By a Trailing Period: 3 days of data at the security resolution:
eth_symbol = self.AddCrypto('ETHUSD', Resolution.Tick).Symbol
ticks = self.History[Tick](eth_symbol, timedelta(days=3))
ticks_df = self.History(eth_symbol, timedelta(days=3))

vix_data = self.History[CBOE](vix_symbol, timedelta(days=3))
trade_bars = self.History[TradeBar](btc_symbol, timedelta(days=3))
quote_bars = self.History[QuoteBar](btc_symbol, timedelta(days=3))
trade_bars_df = self.History(TradeBar, btc_symbol, timedelta(days=3))
quote_bars_df = self.History(QuoteBar, btc_symbol, timedelta(days=3))
df = self.History(btc_symbol, timedelta(days=3))  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-07-08 04:01:00	22141.16	22153.87	22116.38	22123.94	0.400000	22137.01	22153.86	22113.42	22123.93	0.136000	22144.89	22153.87	22116.38	22125.99	30.040683
	2022-07-08 04:02:00	22144.58	22165.07	22135.77	22141.16	0.057960	22144.57	22145.74	22132.86	22137.01	0.907468	22144.58	22164.97	22135.76	22140.51	30.609148
	2022-07-08 04:03:00	22121.26	22146.46	22118.96	22144.58	0.013358	22121.09	22146.45	22118.92	22144.57	0.001288	22121.78	22146.45	22118.96	22144.57	31.833847
	2022-07-08 04:04:00	22114.21	22121.26	22097.83	22121.26	0.002549	22114.20	22121.16	22096.63	22121.09	0.030212	22114.21	22121.18	22097.52	22121.09	10.173673
	2022-07-08 04:05:00	22108.03	22115.72	22097.83	22114.21	1.276832	22108.02	22114.31	22097.82	22114.20	0.000200	22108.02	22114.98	22097.82	22114.98	7.675039

PY

```
# EXAMPLE 4: Requesting By a Trailing Period: 3 days of data with a specific resolution:
trade_bars = self.History[TradeBar](btc_symbol, timedelta(days=3), Resolution.Daily)
quote_bars = self.History[QuoteBar](btc_symbol, timedelta(days=3), Resolution.Minute)
ticks = self.History[Tick](eth_symbol, timedelta(days=3), Resolution.Tick)

trade_bars_df = self.History(TradeBar, btc_symbol, timedelta(days=3), Resolution.Daily)
quote_bars_df = self.History(QuoteBar, btc_symbol, timedelta(days=3), Resolution.Minute)
ticks_df = self.History(eth_symbol, timedelta(days=3), Resolution.Tick)
df = self.History(btc_symbol, timedelta(days=3), Resolution.Hour)  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-07-08 05:00:00	21974.81	22238.28	21954.82	22123.94	0.024746	21972.03	22233.55	21951.59	22123.93	0.000829	21974.74	22235.11	21951.60	22125.99	739.867210
	2022-07-08 06:00:00	21836.71	22058.00	21744.01	21974.81	0.001000	21836.50	22057.99	21744.00	21972.03	0.042814	21836.45	22058.00	21744.00	21972.10	661.371392
	2022-07-08 07:00:00	21821.14	21868.66	21754.91	21836.71	0.009425	21819.25	21866.78	21752.74	21836.50	0.008751	21821.17	21868.41	21752.78	21836.63	501.845020
	2022-07-08 08:00:00	21787.12	21849.87	21713.45	21821.14	0.001000	21786.73	21848.39	21709.80	21819.25	0.001000	21787.95	21849.87	21709.80	21821.47	566.432573
	2022-07-08 09:00:00	21425.18	21812.07	21415.14	21787.12	0.135118	21423.23	21810.77	21415.13	21786.73	0.000078	21425.90	21810.77	21415.14	21786.73	583.159373

```
# Important Note: Period history requests are relative to "now" algorithm time.
```

PY

```
# EXAMPLE 5: Requesting By a Defined Period: 3 days of data at the security resolution:
start_time = datetime(2022, 1, 1)
end_time = datetime(2022, 1, 4)

vix_data = self.History[CBOE](vix_symbol, start_time, end_time)
trade_bars = self.History[TradeBar](btc_symbol, start_time, end_time)
quote_bars = self.History[QuoteBar](btc_symbol, start_time, end_time)
ticks = self.History[Tick](eth_symbol, start_time, end_time)

trade_bars_df = self.History(TradeBar, btc_symbol, start_time, end_time)
quote_bars_df = self.History(QuoteBar, btc_symbol, start_time, end_time)
ticks_df = self.History(Tick, eth_symbol, start_time, end_time)
df = self.History(btc_symbol, start_time, end_time)  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-01-01 05:01:00	46724.00	46724.00	46682.98	46698.99	0.490830	46723.99	46723.99	46678.61	46696.14	0.002460	46724.00	46724.00	46678.61	46696.45	12.898891
	2022-01-01 05:02:00	46719.51	46742.76	46709.30	46724.00	0.000977	46719.50	46736.09	46708.37	46723.99	0.001225	46719.51	46742.76	46710.29	46723.99	19.193879
	2022-01-01 05:03:00	46713.32	46722.14	46697.04	46719.51	0.223169	46709.32	46719.50	46697.00	46719.50	0.043362	46713.31	46719.50	46697.04	46719.50	3.677076
	2022-01-01 05:04:00	46694.78	46715.84	46690.21	46713.32	0.065120	46694.77	46715.30	46688.76	46709.32	0.021412	46694.78	46715.84	46690.20	46709.33	3.759909
	2022-01-01 05:05:00	46693.46	46706.83	46690.03	46694.78	0.070000	46693.45	46701.72	46690.00	46694.77	0.008974	46693.37	46703.07	46690.00	46694.78	5.703675

PY

```
# EXAMPLE 6: Requesting By a Defined Period: 3 days of data with a specific resolution:
trade_bars = self.History[TradeBar](btc_symbol, start_time, end_time, Resolution.Daily)
quote_bars = self.History[QuoteBar](btc_symbol, start_time, end_time, Resolution.Minute)
ticks = self.History[Tick](eth_symbol, start_time, end_time, Resolution.Tick)

trade_bars_df = self.History(TradeBar, btc_symbol, start_time, end_time, Resolution.Daily)
quote_bars_df = self.History(QuoteBar, btc_symbol, start_time, end_time, Resolution.Minute)
ticks_df = self.History(eth_symbol, start_time, end_time, Resolution.Tick)
df = self.History(btc_symbol, start_time, end_time, Resolution.Hour)  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-01-01 06:00:00	47188.00	47548.53	46669.76	46698.99	0.070150	47187.99	47546.38	46667.81	46696.14	0.254247	47188.00	47548.53	46668.64	46696.45	642.638402
	2022-01-01 07:00:00	46982.83	47329.31	46944.32	47188.00	0.017207	46977.97	47329.30	46944.31	47187.99	0.026489	46976.98	47327.89	46944.81	47188.00	380.560480
	2022-01-01 08:00:00	47200.57	47251.92	46905.43	46982.83	0.100000	47200.01	47251.90	46903.50	46977.97	0.000200	47200.01	47251.76	46903.50	46982.82	298.901283
	2022-01-01 09:00:00	47119.93	47349.54	47087.17	47200.57	0.101920	47119.92	47347.53	47087.16	47200.01	0.000016	47119.93	47347.54	47087.17	47200.14	215.983246
	2022-01-01 10:00:00	47139.67	47213.34	46937.35	47119.93	0.012300	47139.66	47210.63	46937.30	47119.92	0.043536	47139.67	47212.89	46938.37	47124.07	137.015047

Multiple Symbol History Requests

To request history for multiple symbols at a time, pass an array of `Symbol` objects to the same API methods shown in the preceding section. The return type of the method call depends on the history request `[Type]` . The following table describes the return type of each request `[Type]` :

Request Type	Return Data Type
No argument	`DataFrame`
`TradeBar`	`List[TradeBars]`
`QuoteBar`	`List[QuoteBars]`
`Tick`	`List[Ticks]`
`alternativeDataClass` (ex: `CBOE`)	`List[Dict[Symbol, alternativeDataClass ]]` (ex: `List[Dict[Symbol, CBOE]]`)

PY

```
# EXAMPLE 7: Requesting By Bar Count for Multiple Symbols: 2 bars at the security resolution:
vix = self.AddData[CBOE]("VIX", Resolution.Daily).Symbol
v3m = self.AddData[CBOE]("VIX3M", Resolution.Daily).Symbol
cboe_data = self.History[CBOE]([vix, v3m], 2)

ibm = self.AddEquity("IBM", Resolution.Minute).Symbol
aapl = self.AddEquity("AAPL", Resolution.Minute).Symbol
trade_bars_list = self.History[TradeBar]([ibm, aapl], 2)
quote_bars_list = self.History[QuoteBar]([ibm, aapl], 2)

trade_bars_df = self.History(TradeBar, [ibm, aapl], 2)
quote_bars_df = self.History(QuoteBar, [ibm, aapl], 2)
df = self.History([ibm, aapl], 2)  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
IBM R735QTJ8XC9X	2022-07-08 15:59:00	140.50	140.63	140.45	140.62	500.0	140.48	140.62	140.44	140.61	100.0	140.49	140.61	140.45	140.61	39645.0
	2022-07-08 16:00:00	140.50	140.51	140.39	140.50	2200.0	140.48	140.50	140.35	140.48	500.0	140.47	140.50	140.35	140.48	542791.0
AAPL R735QTJ8XC9X	2022-07-08 15:59:00	147.03	147.08	146.95	147.07	700.0	147.02	147.07	146.94	147.06	1300.0	147.02	147.08	146.94	147.06	500958.0
	2022-07-08 16:00:00	147.04	147.05	146.86	147.03	7100.0	147.03	147.03	146.84	147.02	300.0	147.04	147.06	146.84	147.02	4307608.0

PY

```
# EXAMPLE 8: Requesting By Bar Count for Multiple Symbols: 5 bars with a specific resolution:
trade_bars_list = self.History[TradeBar]([ibm, aapl], 5, Resolution.Daily)
quote_bars_list = self.History[QuoteBar]([ibm, aapl], 5, Resolution.Minute)

trade_bars_df = self.History(TradeBar, [ibm, aapl], 5, Resolution.Daily)
quote_bars_df = self.History(QuoteBar, [ibm, aapl], 5, Resolution.Minute)
df = self.History([ibm, aapl], 5, Resolution.Daily)  # Includes trade data only. No quote for daily equity data
```

		close	high	low	open	volume
symbol	time					
IBM R735QTJ8XC9X	2022-07-08	140.83	141.32	138.83	139.07	3834112.0
	2022-07-09	140.47	141.32	139.82	140.76	2763286.0
AAPL R735QTJ8XC9X	2022-07-08	146.35	146.55	143.24	143.36	62554270.0
	2022-07-09	147.04	147.55	145.00	145.13	61610642.0

PY

```
# EXAMPLE 9: Requesting By Trailing Period: 3 days of data at the security resolution:
ticks = self.History[Tick]([eth_symbol], timedelta(days=3))

trade_bars = self.History[TradeBar]([btc_symbol], timedelta(days=3))
quote_bars = self.History[QuoteBar]([btc_symbol], timedelta(days=3))
trade_bars_df = self.History(TradeBar, [btc_symbol], timedelta(days=3))
quote_bars_df = self.History(QuoteBar, [btc_symbol], timedelta(days=3))
df = self.History([btc_symbol], timedelta(days=3))  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-07-08 04:01:00	22141.16	22153.87	22116.38	22123.94	0.400000	22137.01	22153.86	22113.42	22123.93	0.136000	22144.89	22153.87	22116.38	22125.99	30.040683
	2022-07-08 04:02:00	22144.58	22165.07	22135.77	22141.16	0.057960	22144.57	22145.74	22132.86	22137.01	0.907468	22144.58	22164.97	22135.76	22140.51	30.609148
	2022-07-08 04:03:00	22121.26	22146.46	22118.96	22144.58	0.013358	22121.09	22146.45	22118.92	22144.57	0.001288	22121.78	22146.45	22118.96	22144.57	31.833847
	2022-07-08 04:04:00	22114.21	22121.26	22097.83	22121.26	0.002549	22114.20	22121.16	22096.63	22121.09	0.030212	22114.21	22121.18	22097.52	22121.09	10.173673
	2022-07-08 04:05:00	22108.03	22115.72	22097.83	22114.21	1.276832	22108.02	22114.31	22097.82	22114.20	0.000200	22108.02	22114.98	22097.82	22114.98	7.675039

PY

```
# EXAMPLE 10: Requesting By Defined Period: 3 days of data at the security resolution:
trade_bars = self.History[TradeBar]([btc_symbol], start_time, end_time)
quote_bars = self.History[QuoteBar]([btc_symbol], start_time, end_time)
ticks = self.History[Tick]([eth_symbol], start_time, end_time)
trade_bars_df = self.History(TradeBar, btc_symbol, start_time, end_time)
quote_bars_df = self.History(QuoteBar, btc_symbol, start_time, end_time)
ticks_df = self.History(Tick, eth_symbol, start_time, end_time)
df = self.History([btc_symbol], start_time, end_time)  # Includes trade and quote data
```

		askclose	askhigh	asklow	askopen	asksize	bidclose	bidhigh	bidlow	bidopen	bidsize	close	high	low	open	volume
symbol	time															
BTCUSD XJ	2022-01-01 05:01:00	46724.00	46724.00	46682.98	46698.99	0.490830	46723.99	46723.99	46678.61	46696.14	0.002460	46724.00	46724.00	46678.61	46696.45	12.898891
	2022-01-01 05:02:00	46719.51	46742.76	46709.30	46724.00	0.000977	46719.50	46736.09	46708.37	46723.99	0.001225	46719.51	46742.76	46710.29	46723.99	19.193879
	2022-01-01 05:03:00	46713.32	46722.14	46697.04	46719.51	0.223169	46709.32	46719.50	46697.00	46719.50	0.043362	46713.31	46719.50	46697.04	46719.50	3.677076
	2022-01-01 05:04:00	46694.78	46715.84	46690.21	46713.32	0.065120	46694.77	46715.30	46688.76	46709.32	0.021412	46694.78	46715.84	46690.20	46709.33	3.759909
	2022-01-01 05:05:00	46693.46	46706.83	46690.03	46694.78	0.070000	46693.45	46701.72	46690.00	46694.77	0.008974	46693.37	46703.07	46690.00	46694.78	5.703675

If you request data for multiple securities and you use the `Tick` request type, each `Ticks` object in the list of results only contains the last tick of each security for that particular timeslice .

All Symbol History Requests

You can request history for all active securities in your universe. The parameters are very similar to other history method calls, but the return type is an array of Slice objects. The `Slice` object holds all of the results in a sorted enumerable collection that you can iterate over with a loop.

PY

```
# EXAMPLE 11: Requesting 5 bars for all securities at their respective resolution:

# Create subscriptions
self.AddEquity("IBM", Resolution.Daily)
self.AddEquity("AAPL", Resolution.Daily)

# Request history data and enumerate results
slices = self.History(5)
for s in slices:
    self.Log(str(s.Time) + " AAPL:" + str(s.Bars["AAPL"].Close) + " IBM:" + str(s.Bars["IBM"].Close))
```

```
2022-07-02 00:00:00 AAPL:138.93 IBM:141.12
2022-07-06 00:00:00 AAPL:141.56 IBM:137.62
2022-07-07 00:00:00 AAPL:142.92 IBM:138.08
2022-07-08 00:00:00 AAPL:146.35 IBM:140.83
2022-07-09 00:00:00 AAPL:147.04 IBM:140.47
```

PY

```
# EXAMPLE 12: Requesting 5 minutes for all securities:

slices = self.History(timedelta(minutes=5), Resolution.Minute)
for s in slices:
    self.Log(str(s.Time) + " AAPL:" + str(s.Bars["AAPL"].Close) + " IBM:" + str(s.Bars["IBM"].Close))
```

```
2022-07-08 15:56:00 AAPL:147.2 IBM:140.71
2022-07-08 15:57:00 AAPL:147.125 IBM:140.67
2022-07-08 15:58:00 AAPL:147.065 IBM:140.61
2022-07-08 15:59:00 AAPL:147.02 IBM:140.49
2022-07-08 16:00:00 AAPL:147.04 IBM:140.47
```

```
# timedelta history requests are relative to "now" in algorithm Time. If you request this data at 16:05,
it returns an empty array because the market is closed.
```

Assumed Default Values

The following table describes the assumptions of the History API:

Argument	Assumption
Resolution	LEAN guesses the resolution you request by looking at the securities you already have in your algorithm. If you have a security subscription in your algorithm with a matching `Symbol` , the history request uses the same resolution as the subscription. If you don't have a security subscription in your algorithm with a matching `Symbol` , `Resolution.Minute` is the default.

Additional Options

The `History` method accepts the following additional arguments:

Argument	Data Type	Description	Default Value
`fillForward`	`bool/NoneType`	True to fill forward missing data. Otherwise, false.	`None`
`extendedMarketHours`	`bool/NoneType`	True to include extended market hours data. Otherwise, false.	`None`
`dataMappingMode`	`DataMappingMode/NoneType`	The contract mapping mode to use for the security history request.	`None`
`dataNormalizationMode`	`DataNormalizationMode/NoneType`	The price scaling mode to use for US Equities or continuous Futures contracts . If you don't provide a value, it uses the data normalization mode of the security subscription.	`None`
`contractDepthOffset`	`int/NoneType`	The desired offset from the current front month for continuous Futures contracts .	`None`

```py
self.future = self.AddFuture(Futures.Currencies.BTC)
history = self.History(
    tickers=[self.future.Symbol],
    start=self.Time - timedelta(days=15),
    end=self.Time,
    resolution=Resolution.Minute,
    fillForward=False,
    extendedMarketHours=False,
    dataMappingMode=DataMappingMode.OpenInterest,
    dataNormalizationMode=DataNormalizationMode.Raw,
    contractDepthOffset=0)
```

Analyze Results

The History method returns a multi-index DataFrame where the first index is the `Symbol` . The data is then sorted in rows according to the second index, the EndTime of the bar. By learning a few helpful shortcuts, you can directly access the history values you need for your algorithm.

PY

```
# Setup the universe:
eurusd = self.AddForex("EURUSD", Resolution.Daily).Symbol
nzdusd = self.AddForex("NZDUSD", Resolution.Daily).Symbol

# STEP 1:  Request a DataFrame:
self.df = self.History([eurusd, nzdusd], 3)
```

		askclose	askhigh	asklow	askopen	bidclose	bidhigh	bidlow	bidopen	close	high	low	open
symbol	time												
EURUSD	2019-04-25	1.11586	1.12247	1.11418	1.12173	1.11562	1.12223	1.11394	1.12150	1.115740	1.122350	1.114060	1.121615
	2019-04-26	1.11364	1.11637	1.11193	1.11585	1.11341	1.11614	1.11169	1.11562	1.113525	1.116255	1.111810	1.115735
	2019-04-27	1.11458	1.11751	1.11117	1.11365	1.11381	1.11729	1.11086	1.11341	1.114195	1.117400	1.111015	1.113530
NZDUSD	2019-04-25	0.65963	0.66316	0.65830	0.66234	0.65939	0.66294	0.65807	0.66210	0.659510	0.663050	0.658185	0.662220
	2019-04-26	0.66447	0.66469	0.65815	0.65964	0.66423	0.66447	0.65790	0.65939	0.664350	0.664580	0.658025	0.659515
	2019-04-27	0.66598	0.66832	0.66370	0.66447	0.66549	0.66810	0.66311	0.66424	0.665735	0.668210	0.663405	0.664355

```
# STEP 2: Check if the DataFrame is empty and lock onto a Symbol index with the `loc[]` method.
#  You can access a security's data in a DataFrame by using the Symbol or the string representation of the Symbol
if not self.df.empty:
    eurusd_quotebars = self.df.loc[eurusd] # or just "EURUSD" instead of Symbol
```

	askclose	askhigh	asklow	askopen	bidclose	bidhigh	bidlow	bidopen	close	high	low	open
time												
2019-04-23	1.12512	1.12633	1.12385	1.12395	1.12488	1.12609	1.12361	1.12372	1.125000	1.126210	1.123730	1.123835
2019-04-24	1.12173	1.12625	1.11935	1.12512	1.12149	1.12602	1.11911	1.12489	1.121610	1.126135	1.119230	1.125005
2019-04-25	1.11586	1.12247	1.11418	1.12173	1.11562	1.12223	1.11394	1.12150	1.115740	1.122350	1.114060	1.121615
2019-04-26	1.11364	1.11637	1.11193	1.11585	1.11341	1.11614	1.11169	1.11562	1.113525	1.116255	1.111810	1.115735
2019-04-27	1.11458	1.11751	1.11117	1.11365	1.11381	1.11729	1.11086	1.11341	1.114195	1.117400	1.111015	1.113530

```
# STEP 3: Extract and manipulate a single column with the string column name
spread = eurusd_quotebars["askclose"] - eurusd_quotebars["bidclose"]
```

```
time
2019-04-23    0.00024
2019-04-24    0.00024
2019-04-25    0.00024
2019-04-26    0.00023
2019-04-27    0.00077
dtype: float64
```

```
# Make sure to use the lowercase string column name.
```

To perform analysis between assets, you can unstack the DataFrame . The `unstack` method transforms each column into the `Symbol` values for one of the price-columns you select.

```
# UNSTACKING: Transform into columns

# Fetch multi-indexed history:
self.dataframe = self.History([self.Symbol("IBM"), self.Symbol("AAPL")], 3)
```

		close	high	low	open	volume
symbol	time					
IBM	2019-04-25	139.98	141.29	139.81	140.60	1231415.0
	2019-04-26	138.62	139.82	137.70	139.82	1419038.0
	2019-04-27	139.44	139.89	138.81	139.28	1275702.0
AAPL	2019-04-25	207.12	208.48	207.05	207.34	8788539.0
	2019-04-26	205.24	207.76	205.12	206.84	10105760.0
	2019-04-27	204.29	205.00	202.12	204.83	9984854.0

```
# Transform using unstack:
self.dataframe["close"].unstack(level=0)
```

symbol	AAPL	IBM
time		
2019-04-25	207.12	139.98
2019-04-26	205.24	138.62
2019-04-27	204.29	139.44

```
# Make sure to use the lowercase string column name.
```

Data Formats

We provide a specific data formats for each asset class. To view the available data formats, see the **Resolutions** documentation of the following asset classes:

- US Equity
- India Equity
- Equity Options
- Crypto
- Crypto Futures
- Forex
- Futures
- Future Options
- Index
- Index Options
- CFD

Common Errors

Errors can occur when you request historical data.

Empty Data Errors

If the history request returns an empty DataFrame and you try to slice it, it throws an exception. To avoid issues, check if the DataFrame contains data before slicing it.

```
df = self.History(symbol, 10).close    # raises exception if the request is empty

def GetSafeHistoryCloses(self, symbols):
    if not symbols:
        self.Log(f'No symbols')
        return  False, None
    df = self.History(symbols, 100, Resolution.Daily)
    if df.empty:
        self.Log(f'Empy history for {symbols}')
        return  False, None
     return True, df.close.unstack(0)
```

If you run algorithms on your local machine and history requests return no data, check if your **data** directory contains the data you request. To download datasets, see Download .

Numerical Precision Errors

Some factor files have INF split values, which indicate that the stock has so many splits that prices can't be calculated with correct numerical precision. To allow history requests with these symbols, we need to move the starting date forward when reading the data. If there are numerical precision errors in the factor files for a security in your history request, LEAN throws the following error:

"Warning: when performing history requests, the start date will be adjusted if there are numerical precision errors in the factor files."

Live Trading Considerations

In live trading, if you make a history request for minute data at noon and the history period covers the start of the previous day to the present moment, the data from the previous day will be backtest data. The data of the current day will be live data that we collected throughout the morning. If you make this history request in a backtest, you might get slightly different data for the current day because of post-processing from the data vendor.

Examples

Demonstration Algorithm
HistoryAlgorithm.py Python WarmupHistoryAlgorithm.py Python

Historical Data

Warm Up Periods

Introduction

LEAN supports an automated fast-forward system called "Warm Up". It simulates winding back the clock from the time you deploy the algorithm. In a backest, this is the `StartDate` of your algorithm. In live trading, it's the current date. Warm Up is a great way to prepare your algorithm and its indicators for trading.

Set Warm Up Periods

You can set a warm-up period based on a period of time or a number of bars. To set a warm-up, in your algorithm's `Initialize` method, call the `SetWarmUp` method.

```
PY
# Wind time back 7 days from start
self.SetWarmUp(timedelta(7))

# Feed in 100 bars before start date
self.SetWarmUp(100)
```

If you pass a `timedelta` argument, the warm-up period consists of that `timedelta` before the start date and pipes in data that matches the resolution of your data subscriptions.

If you pass just an integer argument, LEAN calculates the start of the warm-up period for each of your security subscriptions by using the number of bars you specify, the resolution of each security, and the trading calendar of each security. After it calculates the start time of the warm-up period for each security, it sets the earliest start time as the start of the algorithm warm-up period. For instance, in the following example, the warm-up period consists of 100 minute resolution bars for SPY and as many second resolution bars for BTCUSD that occur from the start of the SPY warm-up period to the algorithm `StartDate` .

```
PY
self.AddEquity("SPY", Resolution.Minute, fillForward=False)
self.AddCrypto("BTCUSD", Resolution.Second, fillForward=False)
self.SetWarmUp(100)
```

To use a specific resolution of data for the warm-up period, pass a `resolution` argument to the `SetWarmUp` method.

```
PY
# Feed in 100 days of daily data before the start date
self.SetWarmUp(timedelta(days=100), Resolution.Daily)

# Feed in data for 100 trading days before the start date
self.SetWarmUp(100, Resolution.Daily)
```

If you pass a `timedelta` and a resolution, the warm-up period consists of the time period and resolution you set, regardless of the resolution of your data subscriptions.

If you pass an integer and a resolution, the warm-up period consists of the number of bars and resolution you set, regardless of the resolution of your data subscriptions. In this case, LEAN calculates the start of the warm-up period for each of your security subscriptions just like it does when you only pass an integer argument. After it calculates the start time of the warm-up period for each security, it sets the earliest start time as the start of the algorithm warm-up period.

If you pass a resolution argument and you registered indicators or consolidators for automatic updates, the warm-up period resolution must be less than or equal to the lowest resolution of your automatic indicators and consolidators. For instance, in the following example, the indicators use minute resolution data, so you can set the warm-up period to use tick, second, or minute resolution data.

```
PY

self.spy = self.AddEquity("SPY", Resolution.Minute, fillForward=False).Symbol
self.btc = self.AddCrypto("BTCUSD", Resolution.Second, fillForward=False).Symbol

self.spy_sma = self.SMA(self.spy, 10)
self.btc_sma = self.SMA(self.btc, 10, Resolution.Minute)

self.SetWarmUp(100, Resolution.Minute)
```

Warm Up Vs Reality

You may need to distinguish warm-up data from real data. To check if the algorithm is currently in a warm up period, use the `IsWarmingUp` property.

```
PY

# In Initialize
self.emaFast = self.EMA("IBM", 50);
self.emaSlow = self.EMA("IBM", 100);
self.SetWarmup(100);

// In OnData: Don't run if the indicators aren't ready
if self.IsWarmingUp: return
```

Event Handler

The `OnWarmupFinished` event handler executes when the warm up period ends, even if you don't set a warm up period.

```
PY

def OnWarmupFinished(self) -> None:
    pass # Done warming up
```

Limitations

You can't place trades during the warm-up period because they would be operating on fictional fast-forwarded data.

Examples

Demonstration Algorithms
WarmupAlgorithm.py Python

Historical Data

Rolling Window

Introduction

A `RollingWindow` is an array of a fixed-size that holds trailing data. It's more efficient to use `RollingWindow` objects to hold periods of data than to make multiple historical data requests . With a `RollingWindow` , you just update the latest data point while a `History` call fetches all of the data over the period you request. `RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window.

Supported Types

`RollingWindow` objects can store any native or C# types.

```
PY
self.close_window = RollingWindow[float](4)
self.trade_bar_window = RollingWindow[TradeBar](2)
self.quote_bar_window = RollingWindow[QuoteBar](2)
```

To be notified when `RollingWindow` objects support additional types, subscribe to GitHub Issue #6199 .

Add Data

To add data to a `RollingWindow` , call the `Add` method.

```
PY
self.close_window.Add(data["SPY"].Close)
self.trade_bar_window.Add(data["SPY"])
self.quote_bar_window.Add(data["EURUSD"])
```

Warm Up

To warm up a `RollingWindow` , make a history request and then iterate through the result to add the data to the `RollingWindow` .

```
PY
spy = self.AddEquity("SPY", Resolution.Daily).Symbol
history_trade_bar = self.History[TradeBar](spy, 10, Resolution.Daily)
history_quote_bar = self.History[QuoteBar](spy, 10, Resolution.Minute)

# Warm up the close price and trade bar rolling windows with the previous 10-day trade bar data
close_price_window = RollingWindow[float](10)
trade_bar_window = RollingWindow[TradeBar](10)
for trade_bar in history_trade_bar:
    close_price_window.Add(trade_bar.Close)
    trade_bar_window.Add(trade_bar)

# Warm up the quote bar rolling window with the previous 10-minute quote bar data
quote_bar_window = RollingWindow[QuoteBar](10)
for quote_bar in history_quote_bar:
    quote_bar_window.Add(quote_bar)
```

Check Readiness

To check if a `RollingWindow` is full, use its `IsReady` flag.

```
PY
if not self.close_window.IsReady:
    return
```

Access Data

`RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window.

```
PY
current_close = self.close_window[0]
previous_close = self.close_window[1]
oldest_close = self.close_window[self.close_window.Count-1]
```

To get the item that was most recently removed from the `RollingWindow` , use the `MostRecentlyRemoved` property.

```
PY
removed_close = self.close_window.MostRecentlyRemoved
```

Combine with Indicators

To track historical indicator values, use a `RollingWindow` . Indicators emit an `Updated` event when they update. To create a `RollingWindow` of indicator points, attach an event handler function to the `Updated` member that adds the last value of the indicator to the `RollingWindow` . The value is an `IndicatorDataPoint` object that represents a piece of data at a specific time.

```
PY
def Initialize(self) -> None:
    # Creates an indicator and adds to a RollingWindow when it is updated
    self.sma_window = RollingWindow[IndicatorDataPoint](5)
    self.SMA("SPY", 5).Updated += (lambda sender, updated: self.sma_window.Add(updated))
```

To view how to access individual members in an indicator, see Get Indicator Values .

```
PY
current_sma = self.sma_window[0]
previous_sma = self.sma_window[1]
oldest_sma = self.sma_window[sma_window.Count - 1]
```

Combine with Consolidators

To store a history of consolidated bars , in the consolidation handler, add the consolidated bar to the `RollingWindow` .

```
PY
self.consolidator.DataConsolidated += lambda sender, consolidated_bar:
self.trade_bar_window.Add(consolidated_bar)
```

Cast to Other Types

You can cast a `RollingWindow` to a list or a DataFrame. If you cast it to a list, reverse the list so the most recent element is at the last index of the list. This is the order the elements would be in if you added the elements to the list with the `Add` method. To cast a `RollingWindow` to a DataFrame, the `RollingWindow` must contain `Slice` , `Tick` , `QuoteBar` , or `TradeBar` objects. If the `RollingWindow` contains ticks, the ticks must have unique timestamps.

```
PY

closes = list(self.close_window)[::-1]

tick_df = self.PandasConverter.GetDataFrame[Tick](list(self.tick_window)[::-1])
trade_bar_df = self.PandasConverter.GetDataFrame[TradeBar](list(self.trade_bar_window)[::-1])
quote_bar_df = self.PandasConverter.GetDataFrame[QuoteBar](list(self.quote_bar_window)[::-1])
```

Delete Data

To remove all of the elements from a `RollingWindow` , call the `Reset` method.

```
PY

self.close_window.Reset()
```

Examples

Demonstration Algorithm
RollingWindowAlgorithm.py Python CustomVolatilityModelAlgorithm.py Python
MultipleSymbolConsolidationAlgorithm.py Python

Trading and Orders

Trading and Orders

Key Concepts

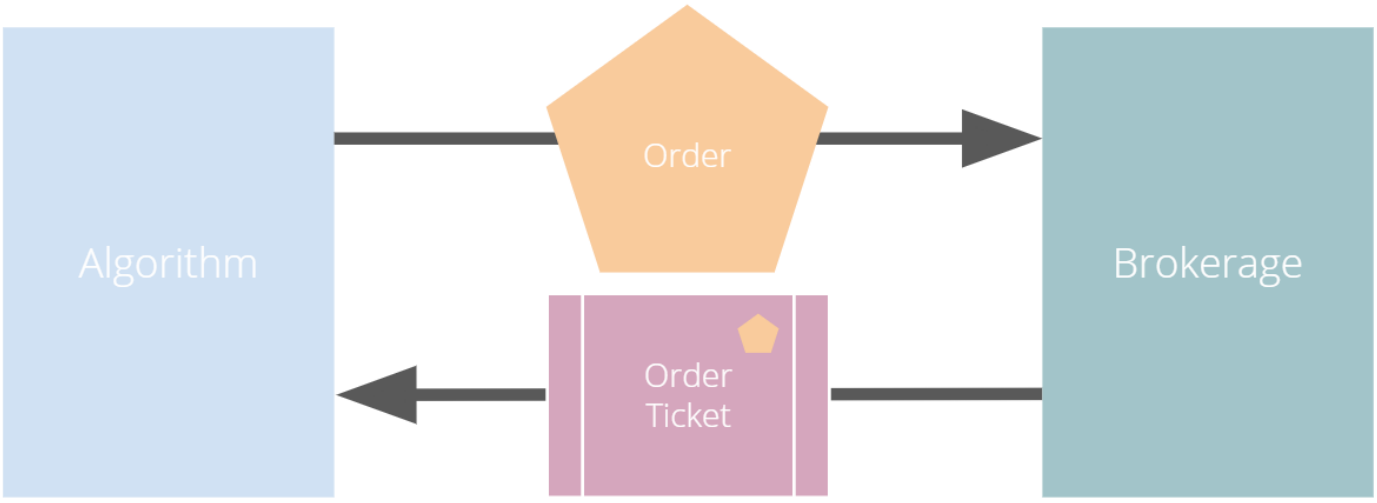
Introduction

LEAN has dozens of methods to create, update, and cancel orders. You can place orders automatically with helper methods or manually through methods on the algorithm API. You can fetch, update, and cancel manual orders with order tickets . As the state of your orders change, LEAN creates events that notify your algorithm.

In backtesting, LEAN simulates order fills with historical data, but you can create your own fill, fee, slippage, and margin models via plugin points. You control how optimistic or pessimistic order fills are with transaction model classes. For more information about these types of models, see Reality Modeling .

Orders and Tickets

When you call one of the order methods to place an order, LEAN performs pre-order checks to ensure that the order meets some requirements and uses the last known price to check you have enough capital . To see the requirements of each order, see the **Requirements** section of the documentation for the order types and the **Orders** section of your brokerage model . If you can place the order, LEAN creates `Order` and OrderTicket objects. The order ticket is sent to your brokerage. As the brokerage processes your order, it returns another order ticket and it's compared against the order to see if the order is satisfied. Orders are asynchronous in live trading, so if you want to change an order, you must request it with the order ticket. Order changes are not guaranteed since your order may fill by the time brokerage receives the request.



Order Types

LEAN supports the following order types:

- Market
- Limit
- Limit if touched
- Stop market
- Stop limit
- Market on open
- Market on close
- Combo market
- Combo limit
- Combo leg limit
- Option exercise

Symbol Properties

The `SymbolProperties` are a property of the `Security` object. LEAN uses some of the `SymbolProperties` to prevent invalid orders, and to calculate order quantities for a given target.

`SymbolProperties` objects have the following properties:

To get the `SymbolProperties` , use the property on the `Security` object.

```py
symbol_properties = self.Securities["BTCUSD"].SymbolProperties
lot_size = symbol_properties.LotSize
minimum_order_size = symbol_properties.MinimumOrderSize
minimum_price_variation = symbol_properties.MinimumPriceVariation
```

LEAN uses the `MinimumPriceVariation` to round the `LimitPrice` , `StopPrice` , and the `TriggerPrice` .

Quote Currency

The quote currency is the currency you must give the seller to buy an asset. For currency trades, the quote currency is the currency ticker on the right side of the currency pair. For other types of assets, the quote currency is usually USD, but the quote currency for India Equities is INR. To get the quote currency of a `Security` , use the `QuoteCurrency` property.

```py
aapl_quote_currency = self.Securities["AAPL"].QuoteCurrency # USD
btcusdt_quote_currency = self.Securities["BTCUSDT"].QuoteCurrency # USDT
```

The `QuoteCurrency` is a `Cash` object, which have the following properties:

You can use the `ConversionRate` property to calculate the value of the minimum price movement in the account currency

```
cfd = self.Securities["SG30SGD"]
quote_currency = cfd.QuoteCurrency # SGD
contract_mutiplier = cfd.SymbolProperties.ContractMultiplier
minimum_price_variation = cfd.SymbolProperties.MinimumPriceVariation

# Value of a pip in account currency
pip = minimum_price_variation * contract_mutiplier * quote_currency.ConversionRate
```

Trade Models

Reality models make backtests as realistic as possible to how the strategy would perform in live trading.

Split Adjustment

If a split event occurs before your order is filled, the unfilled portion of the order is adjusted automatically, where its quantity is multiplied by the split factor and the limit/stop/trigger price (if any) is divided by the split factor.

Live Trading Considerations

In live trading, orders fill asynchronously. We send your order to the API of your brokerage and wait for their response to update the state of your algorithm and portfolio. The timing of live order fills doesn't depend on the resolution of your security subscriptions. When your order fills, the fill price and fee is set by your brokerage. You can add event handlers to your algorithm to monitor the brokerage connection and brokerage messages .

In backtesting, the trade fill timing depends on the resolution of your security subscription. For example, if you subscribe to a security with minute resolution data, the algorithm only receives data in one-minute time slices . As a result, the fill model can only evaluate if the order should fill on a minute-by-minute frequency.

Trading and Orders

Order Management

Order Management

Order Tickets

Introduction

When you create an order, you get an `OrderTicket` object to manage your order.

Properties

`OrderTicket` objects have the following attributes:

Track Orders

As the state of your order updates over time, your order ticket automatically updates. To track an order, you can check any of the preceding order ticket properties.

To get an order field, call the `Get` method with an `OrderField` .

```
limit_price = ticket.Get(OrderField.LimitPrice)
```

PY

The `OrderField` enumeration has the following members:

In addition to using order tickets to track orders, you can receive order events through the `OnOrderEvent` event handler.

Update Orders

To update an order, use its `OrderTicket` . You can update other orders until they are filled or the brokerage prevents modifications. You just can't update orders during warm up and initialization .

Updatable Properties

The specific properties you can update depends on the order type. The following table shows the properties you can update for each order type.

Order Type	Updatable Properties				
	Tag	Quantity	LimitPrice	TriggerPrice	StopPrice
Market Order					
Limit Order	✓	✓	✓		
Limit If Touched Order	✓	✓	✓	✓	
Stop Market Order	✓	✓			✓
Stop Limit Order	✓	✓	✓		✓
Market On Open Order	✓	✓			
Market On Close Order	✓	✓			

Update Methods

To update an order, pass an `UpdateOrderFields` object to the `Update` method. The method returns an `OrderResponse` to signal the success or failure of the update request.

```py
 # Create an order
ticket = self.LimitOrder("SPY", 100, 221.05, False, "New SPY trade")

# Update the order tag and limit price
updateSettings = UpdateOrderFields()
updateSettings.LimitPrice = 222.00
updateSettings.Tag = "Limit Price Updated for SPY Trade"
response = ticket.Update(updateSettings)

# Check the OrderResponse
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateStopPrice`
- `UpdateTag`

```py
response = ticket.UpdateLimitPrice(limitPrice, tag)

response = ticket.UpdateQuantity(quantity, tag)

response = ticket.UpdateStopPrice(stopPrice, tag)

response = ticket.UpdateTag(tag)
```

Update Order Requests

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Workaround for Brokerages That Don't Support Updates

Not all brokerages fully support order updates. To check what functionality your brokerage supports for order updates, see the **Orders** section of the documentation for your brokerage model . If your brokerage doesn't support order updates and you want to update an order, cancel the order . When you get an order event that confirms the order is no longer active, place a new order.

```
def OnData(self, slice: Slice) -> None:
    # Cancel the order
    self.ticket.Cancel()

def OnOrderEvent(self, orderEvent: OrderEvent) -> None:
    if self.ticket is not None \
        and orderEvent.OrderId == self.ticket.OrderId \
        and orderEvent.Status == OrderStatus.Canceled:
        # Place a new order
        quantity = self.ticket.Quantity - self.ticket.QuantityFilled
        limit_price = self.Securities[self.ticket.Symbol].Price + 1
        self.ticket = self.LimitOrder(self.ticket.Symbol, quantity, limit_price)
```

Cancel Orders

To cancel an order, call the `Cancel` method on the `OrderTicket` . The method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
 # Create an order and save its ticket
ticket = self.LimitOrder("SPY", 100, 221.05, False, "SPY Trade to Cancel")

# Cancel the order
response = ticket.Cancel("Canceled SPY Trade")

# Use the order response object to read the status
if response.IsSuccess:
    self.Debug("Order successfully canceled")
```

You can't cancel market orders because they are immediately transmitted to the brokerage. You also can't cancel any orders during warm up and initialization .

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Order Response

When you update or cancel an order, LEAN returns an `OrderReponse` object, which have the following attributes:

If your order changes fail, check the `ErrorCode` or `ErrorMessage` . For more information about specific order errors, see the Order Response Error Reference .

To get most recent order response, call the `GetMostRecentOrderResponse` method.

```
response = ticket.GetMostRecentOrderResponse()
```
PY

Examples

Demonstration Algorithms
OrderTicketDemoAlgorithm.py Python

Order Management

Transaction Manager

Introduction

The algorithm transactions manager (`SecurityTransactionManager`) contains a collection of helper methods to quickly access all your orders. To access the transaction manager, use the `Transactions` property of your algorithm. If you save a reference to your order tickets , you shouldn't need to use the transaction manager. LEAN updates the order tickets as the brokerage processes your orders.

Get a Single Order Ticket

If you didn't save a reference to the order ticket when you created an order, you can call the `GetOrderTicket` method to get it. You need to pass the order ID to the method. If you don't have the order ID, you can use the `LastOrderId` property to get the order ID of the most recent order.

```
PY
order_id = self.Transactions.LastOrderId
ticket = self.Transactions.GetOrderTicket(order_id)
```

Get Order Tickets

To get order tickets, call the `GetOrderTickets` or `GetOpenOrderTickets` method. You can pass in a filter function to filter all of the order tickets or pass a `Symbol` to get the order tickets for a specific asset.

```
PY
# Get all order tickets
order_tickets = self.Transactions.GetOrderTickets()

# Get order tickets that pass a filter
filtered_order_tickets = self.Transactions.GetOrderTickets(lambda order_ticket: order_ticket.Symbol ==
symbol)

# Get all open order tickets
open_order_tickets = self.Transactions.GetOpenOrderTickets()

# Get all open order tickets for a symbol
symbol_open_order_tickets = self.Transactions.GetOpenOrderTickets(symbol)

# Get open order tickets that pass a filter
filtered_open_order_tickets = self.Transactions.GetOpenOrderTickets(lambda order_ticket:
order_ticket.Quantity > 10)
```

Get a Single Order

To get a clone of a specific order, call the `GetOrderById` method with the order Id. To get the order Id, use the `OrderId` property of the order ticket or use the `LastOrderID` property if you want the most recent order.

```
PY
order_id = self.Transactions.LastOrderID
order = self.Transactions.GetOrderById(order_id)
```

Order objects are immutable and changes to the order object will not impact the trade. To make an update to an order you must use Order Tickets . `Order` objects have the following attributes:

Get Orders

To get a list of orders, call the `GetOrders` , `GetOpenOrders` , or `GetOrdersByBrokerageId` method. These method returns a list of `Order` objects.

```PY
# Get all completed orders
completed_Orders = self.Transactions.GetOrders()

# Get all completed orders that pass a filter
filtered_completed_orders = self.Transactions.GetOrders(lambda x: x.Quantity > 10)

# Retrieve a list of all completed orders for a symbol
symbol_completed_orders = self.Transactions.GetOrders(lambda x: x.Symbol == symbol)

# Get all open orders
open_orders = self.Transactions.GetOpenOrders()

# Get all open orders that pass a filter
filtered_open_orders = self.Transactions.GetOpenOrders(lambda x: x.Quantity > 10)

# Retrieve a list of all open orders for a symbol
symbol_open_orders = self.Transactions.GetOpenOrders(symbol)

# Get all open and completed orders that correspond to an Id that the brokerage assigned in live trading
orders_by_brokerage_id = self.Transactions.GetOrdersByBrokerageId(brokerageId)
```

`Order` objects have the following attributes:

The `OrdersCount` property gets the current number of orders that have been processed.

Get Remaining Order Quantity

To get the unfilled quantity of open orders, call the `GetOpenOrdersRemainingQuantity` method.

```PY
# Get the quantity of all open orders
all_open_quantity = self.Transactions.GetOpenOrdersRemainingQuantity()

# Get the quantity of open orders that pass a filter
filtered_open_quantity = self.Transactions.GetOpenOrdersRemainingQuantity(
    lambda order_ticket: order_ticket.Quantity > 10
)

# Get the quantity of open orders for a symbol
symbol_open_quantity = self.Transactions.GetOpenOrdersRemainingQuantity(symbol)
```

Cancel Orders

To cancel open orders, call the `CancelOpenOrders` method. This method returns a list of `OrderTicket` objects that correspond to the canceled orders.

```PY
 # Cancel all open orders
all_cancelled_orders = self.Transactions.CancelOpenOrders()

# Cancel orders related to IBM and apply a tag
ibm_cancelled_orders = self.Transactions.CancelOpenOrders("IBM", "Hit stop price")
```

Trading and Orders

Order Types

You can place any of the following order types that your brokerage supports. Click one to learn more.

Market Orders

Limit Orders

Limit if Touched Orders

Stop Market Orders

Stop Limit Orders

Market On Open Orders

Market On Close Orders

Combo Market Orders

Combo Limit Orders

Combo Leg Limit Orders

Option Exercise Orders

Other Order Types

See Also

Brokerages

Order Types

Market Orders

Introduction

Market orders are instructions to trade a specific number of units, regardless of the fill price. If your highest priority is to fill an order as fast as possible, use market orders. If the order book of the security you're trading has sufficient liquidity when the brokerage receives your order, it immediately fills. However, if there is not enough liquidity by the brokerage, you get partial fills and may incur additional costs from market impact.

Place Orders

To send a market order, call the `MarketOrder` , `Buy` , or `Order` method and provide a `Symbol` and quantity. If you don't have sufficient capital for the order, it's rejected. By default, market orders are synchronous and fill immediately.

```py
# Buy orders
self.MarketOrder("IBM", 100)
self.Buy("AAPL", 10)
self.Order("SPY", 20)

# Sell orders
self.MarketOrder("AMZN", -15)
self.Sell("TSLA", 25)
self.Order("SPY", -20)
```

You can also provide a tag and order properties to the `Order` and `MarketOrder` methods.

```py
self.MarketOrder(symbol, quantity, tag=tag, orderProperties=order_properties)
```

If you place a market order during pre-market or post-market hours, LEAN converts your order to market on open order. To override this behavior, create a custom fill model . To be notified when the default fill model allows you to submit market orders outside of regular trading hours, subscribe to GitHub Issue #3947 .

Monitor Order Fills

If the brokerage has sufficient liquidity in their order book, market orders fill immediately. Otherwise, you get partial fills. To monitor the fills of your order, save a reference to the order ticket .

```py
ticket = self.MarketOrder("XLK", 10)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Synchronous Timeouts

Market orders are synchronous by default, so your algorithm waits for the order to fill before moving to the next line of

code. If your order takes longer than five seconds to fill, your algorithm continues executing even if the trade isn't filled. To adjust the timeout period, set the `Transactions.MarketOrderFillTimeout` property.

```
 # Adjust the market fill-timeout to 30 seconds.
self.Transactions.MarketOrderFillTimeout = timedelta(seconds=30)
```

Place Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders, set the `asynchronous` argument to true.

```
self.MarketOrder("IBM", 100, True)
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with market orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for market orders, see the **Orders** section of the brokerage model documentation .

Requirements

Market orders must be submitted during market hours for all security types.

If your algorithm place market orders at or after the last minute of regular market hours, they will be converted into market-on-open orders and will have to observe their for the following asset types:

1. Equities
2. Equity Options
3. Forex
4. CFDs
5. Index Options

Market orders for Futures and Future Options can be submitted during extended market hours, or they will be invalid.

Example

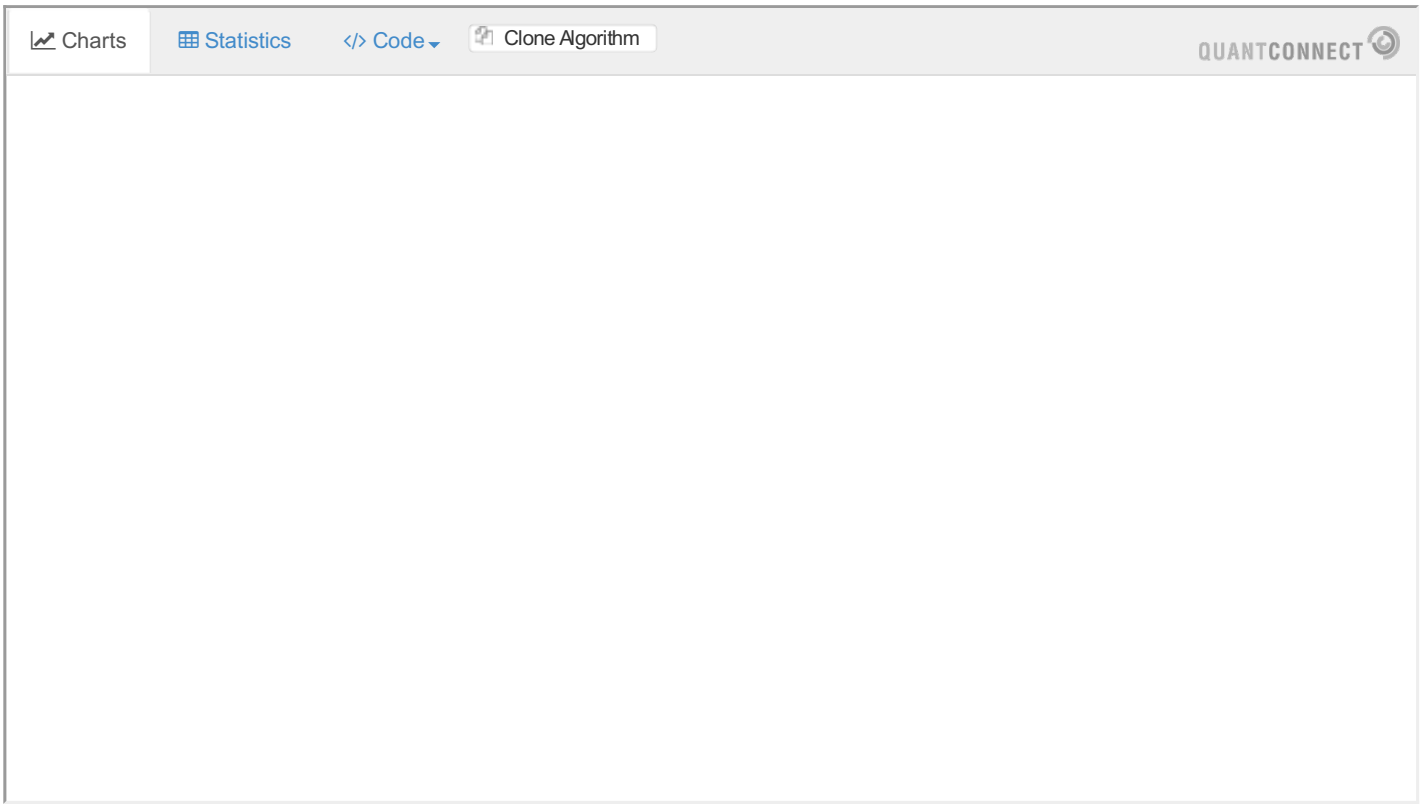
The following backtest verifies the `MarketOrder` behavior. The algorithm buys SPY on the first day and liquidates the position on the second day. The following table shows the first two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	SPY	429.11	10	Market	Filled	4291.10	
2021-07-02T09:31:00Z	SPY	431.54	-10	Market	Filled	-4315.40	

On July 1, 2021, the algorithm buys SPY at $429.11. The fill model fills this order at the ask close price.

On July 2, 2021, the algorithm sells the SPY holdings at $431.54. The fill model fills this order at the bid close price.

To reproduce these results, backtest the following algorithm:



Order Types

Limit Orders

Introduction

Limit orders are instructions to trade a quantity of an asset at a specific price or a better price. A **marketable limit order** has a limit price that crosses the spread. An **unmarketable limit order** has a limit price that doesn't cross the spread. For example, if an asset has an ask price of 100*andyouplacealimitordertobuyat*100 or higher, that's a marketable limit order that should immediately fill. If you place a limit order to buy at $99.99 or lower, that's an unmarketable limit order that should go on the order book. Unmarketable limit orders save you from paying spread costs. Some brokerages even charge you a lower transaction fee if you use unmarketable limit orders rather than marketable limit orders.

Limit orders are helpful in illiquid markets. You can use them to get a good entry price or to set a take-profit level on an existing holding. However, if the market trades away from your limit price, you may have to adjust your limit price or wait for the market to trade back to your limit price to fill the order.

Place Orders

To send a limit order, call the `LimitOrder` method with a `Symbol` , quantity, and limit price. You can also provide a tag and order properties to the `LimitOrder` method. If you do not have sufficient capital for the order, it's rejected.

```
self.LimitOrder(symbol, quantity, limitPrice, tag, orderProperties)
```

PY

To buy an asset with a marketable limit order, set the limit price to the current ask price or higher.

To sell an asset with a marketable limit order, set the limit price to the current bid price or lower.

To buy an asset with an unmarketable limit order, set the limit price below the current ask price.

```
# Buy 1 Bitcoin when the price drops to $34,000
self.LimitOrder("BTCUSD", 1, 34000)
```

PY

To sell an asset with an unmarketable limit order, set the limit price above the current bid price.

```
# Sell 1 Bitcoin when the price moves up to $60,000
self.LimitOrder("BTCUSD", -1, 60000)
```

Monitor Order Fills

Limit orders fill when the security price passes the limit price. To monitor the fills of your order, save a reference to the order ticket .

```
# Buy 10 shares of XLK at $140
ticket = self.LimitOrder("XLK", 10, 140)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, limit price, and tag of limit orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
# Create a new order and save the order ticket
ticket = self.LimitOrder("SPY", 100, 221.05, tag="original tag")

# Update the order
update_settings = UpdateOrderFields()
update_settings.Quantity = 80
update_settings.LimitPrice = 222.00
update_settings.Tag = "new tag"
response = ticket.Update(update_settings)

# Check if the update was successful
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTag`

```
response = ticket.UpdateLimitPrice(limitPrice, tag)

response = ticket.UpdateQuantity(quantity, tag)

response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a limit order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns

`None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with limit orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for limit orders, see the **Orders** section of the brokerage model documentation .

Requirements

Limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Example

The following backtest verifies the `LimitOrder` behavior. On the first day, the algorithm buys SPY with an unmarketable limit order. On the second day, the algorithm sells SPY with an unmarketable limit order and places another unmarketable limit order to buy SPY, which doesn't fill. The following table shows the first three trades in the backtest:

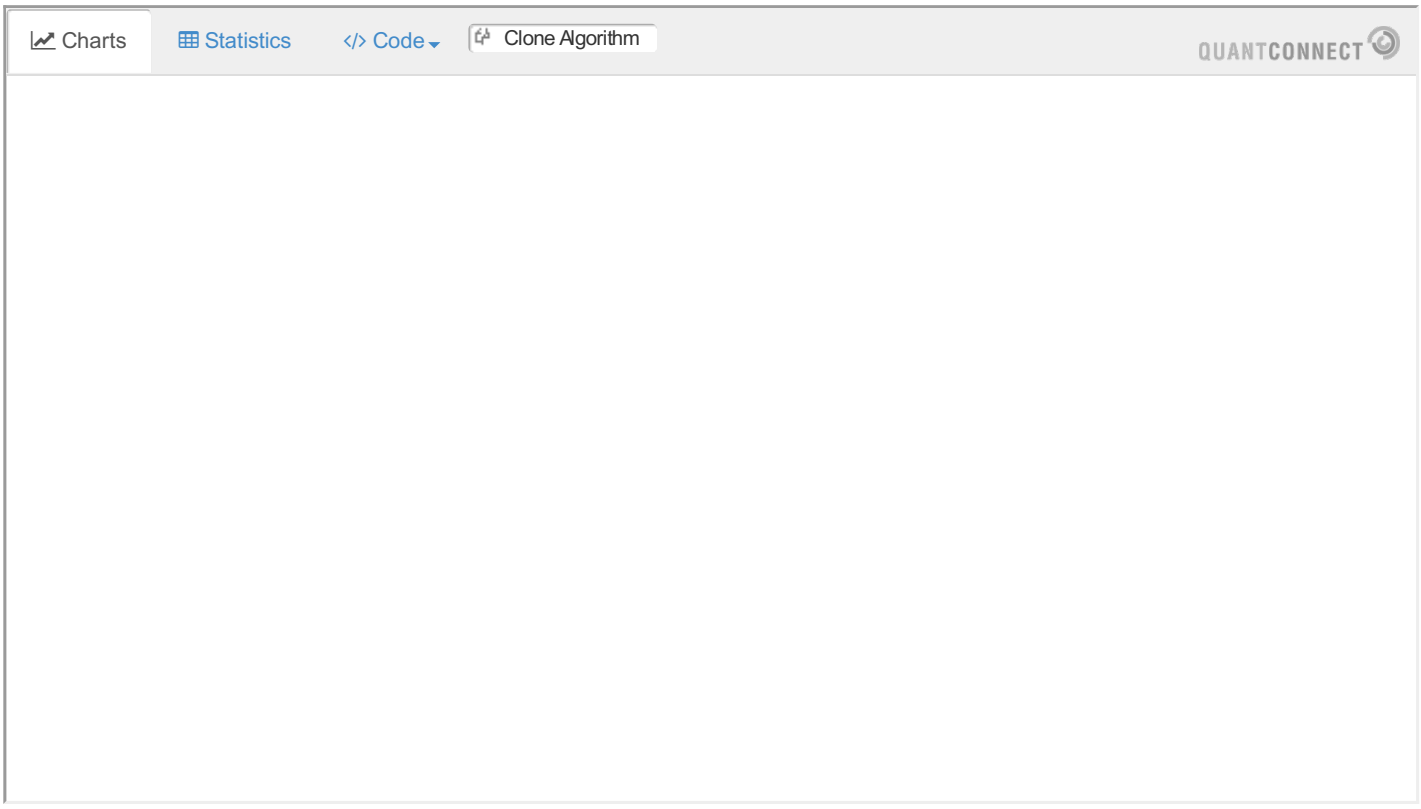
Submitted Time	Filled Time	Symbol	Limit Price	Filled Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	2021-07-01T09:32:00Z	SPY	429.00	429.00	10	Limit	Filled	4290.00	Limit Price: ¤429.00
2021-07-02T09:31:00Z	2021-07-02T09:35:00Z	SPY	431.70	431.70	-10	Limit	Filled	-4317.00	Limit Price: ¤431.70
2021-07-02T09:31:00Z	/	SPY	400.00	/	10	Limit	Submitted	/	Limit Price: ¤400.00

On July 1, 2021 at 9:31 AM Eastern Time (ET), the algorithm places a buy limit order with a limit price of 429*whentheasklowpriceis*428.81 and the ask high price is 429.15.*Theorderfillsat*9:32*AMETatapriceof*429. The fill model fills buy limit orders when the ask low price is less than the limit price. It sets the fill price of the order to the minimum of ask high price and the limit price.

On July 2, 2021 at 9:31 AM ET, the algorithm places a sell limit order at 431.70*andabuylimitorderat*400. At the time of the order, the bid high price is 431.65, *thebidlowpriceis*431.49, and the ask low price is 431.50.*Thesellimitorderfillsat*9:35*AMETatapriceof*431.70 and the buy limit order doesn't fill. The fill model fills sell limit order when the bid high price is greater than the limit price. It sets the fill price of the order to the maximum of the bid

low price and the limit price. The buy limit order doesn't fill because the ask low price is above the limit price for the remainder of the backtest period.

To reproduce these results, backtest the following algorithm:



Order Types

Limit if Touched Orders

Introduction

Limit if touched (LIT) orders are instructions to place a limit order once an asset touches a specific price level. A buy LIT order has a trigger price below the current market price and a sell LIT order has a trigger price above the current market price. In effect, LIT orders are the opposite of stop limit orders .

Place Orders

To send a LIT order, call the `LimitIfTouchedOrder` method and provide a `Symbol` , quantity, trigger price, and limit price. You can also provide a tag and order properties to the `LimitIfTouchedOrder` method. If you do not have sufficient capital for the order, it's rejected.

```
self.LimitIfTouchedOrder(symbol, quantity, trigger_price, limit_price, tag, order_properties)
```

To buy an asset with a LIT order that has a marketable limit price, set the trigger price below the current market price and set the limit price above the trigger price.

```
# Once Bitcoin trades down to $36,000, place a limit order to buy 1 Bitcoin at $38,000
self.LimitIfTouchedOrder("BTCUSD", 1, 33000, 36000)
```

To sell an asset with a LIT order that has a marketable limit price, set the trigger price above the current market price and set the limit price below the trigger price.

```py
# Once Bitcoin trades up to $62,000, place a limit order to sell 1 Bitcoin at $59,000
self.LimitIfTouchedOrder("BTCUSD", -1, 62000, 59000)
```

To buy an asset with a LIT order that has an unmarketable limit price, set the trigger price below the current market price and set the limit price below the trigger price.

```py
# Once Bitcoin trades down to $46,000, place a limit order to buy 1 Bitcoin at $36,000
self.LimitIfTouchedOrder("BTCUSD", 1, 46000, 36000)
```

To sell an asset with a LIT order that has an unmarketable limit price, set the trigger price above the current market price and set the limit price above the trigger price.

```
# Once Bitcoin trades up to $54,000, place a limit order to sell 1 Bitcoin at $62,000
self.LimitIfTouchedOrder("BTCUSD", -1, 54000, 62000)
```

Monitor Order Fills

LIT orders fill when the security price touches the trigger price and is at least as favorable as the limit price. To monitor the fills of your order, save a reference to the order ticket .

```
# Once SPY trades down to $425, place a limit order to buy 10 shares at $415
ticket = self.LimitIfTouchedOrder("SPY", 10, 425, 415)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, trigger price, limit price, and tag of LIT orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
# Create a new order and save the order ticket
ticket = self.LimitIfTouchedOrder("SPY", 100, 350, 340, tag="Original tag")

# Update the order
update_order_fields = UpdateOrderFields()
update_order_fields.Quantity = 80
update_order_fields.TriggerPrice = 380
update_order_fields.LimitPrice = 370
update_order_fields.Tag = "New tag"
response = ticket.Update(update_settings)

# Check the OrderResponse
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTriggerPrice`
- `UpdateTag`

```
response = ticket.UpdateLimitPrice(limitPrice, tag)
response = ticket.UpdateQuantity(quantity, tag)
response = ticket.UpdateTriggerPrice(triggerPrice, tag)
response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a LIT order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with LIT orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for LIT orders, see the **Orders** section of the brokerage model documentation .

Requirements

LIT orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Example

The following backtest verifies the `LimitIfTouchedOrder` behavior. The following table shows the trades in the backtest:

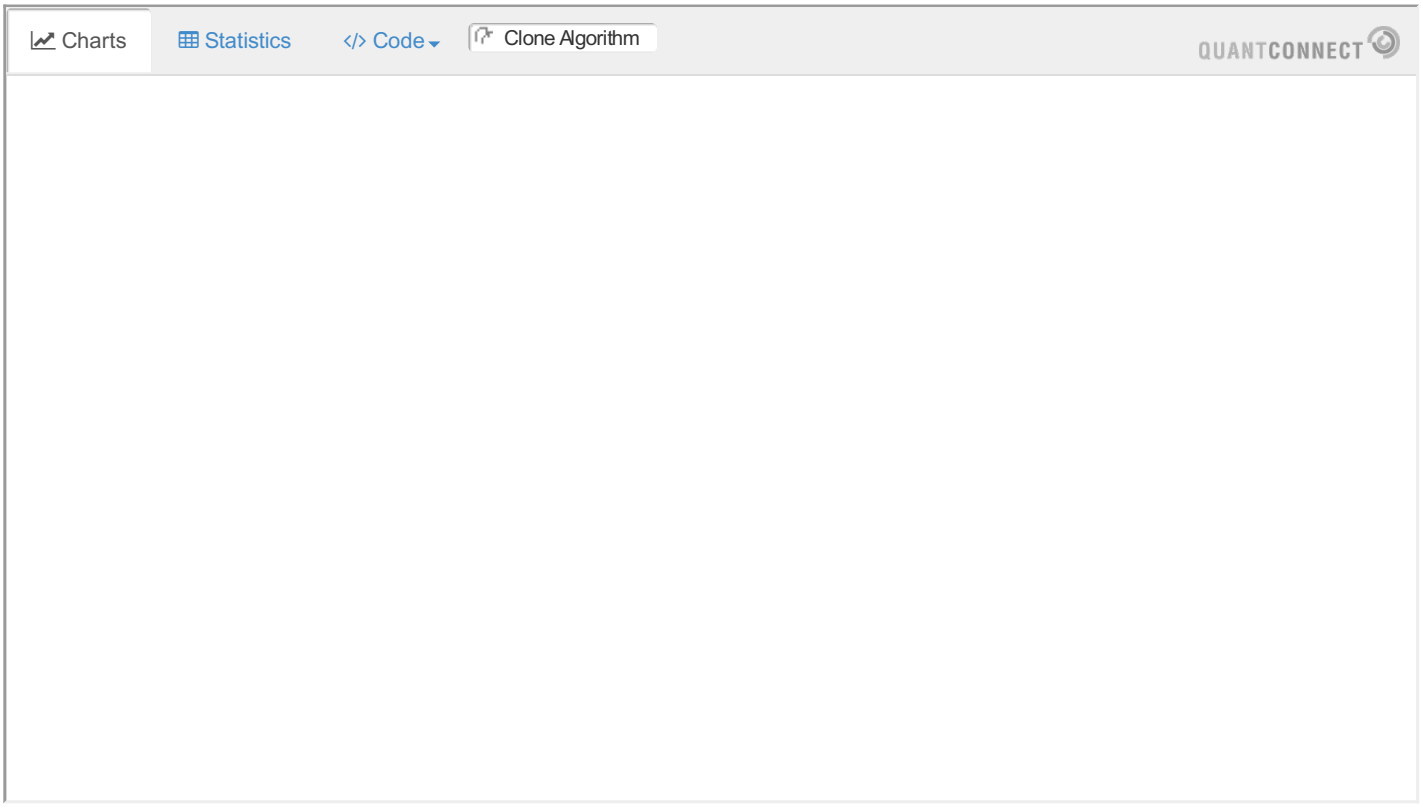
Submitted Time	Filled Time	Symbol	Trigger Price	Limit Price	Filled Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	2021-07-01T09:38:00Z	SPY	429.00	428.95	428.95	10	Limit if touched	Filled	4285.00	Trigger Price: ¤429.00 Limit Price: ¤428.95
2021-07-02T09:31:00Z	2021-07-02T09:49:00Z	SPY	431.70	431.75	431.84	-10	Limit if touched	Filled	-4318.40	Trigger Price: ¤431.70 Limit Price: ¤431.75
2021-07-02T09:31:00Z	/	SPY	431.70	400.00	/	10	Limit if touched	Submitted	/	Trigger Price: ¤431.70 Limit Price: ¤400.00
/	/	SPY	400.00	400.00	/	10	Limit if touched	Submitted	/	Trigger Price: ¤400.00 Limit Price: ¤400.00

On July 1, 2021 at 9:31 AM Eastern Time (ET), the algorithm places a LIT order to buy SPY with a trigger price of 429*andalimitpriceof*428.95. At the time of the order submission, the low price was 428.80*andthebidclosepricewas*429.10. The order fills at 9:38 AM ET at a price of $428.95. The fill model sets the limit price for buy orders when the low price is less than or equal to the trigger price, fills the order when the bid close price is less than or equal to the limit price, and sets the fill price to the minimum of the ask price and the limit price.

On July 2, 2021 at 9:31 AM ET, the algorithm places a LIT order to sell SPY with a trigger price of 431.70*andalimitpriceof* 431.75. At the time of the order submission, the high price was 431.78*andtheaskclosepricewas*431.55. The order fills at 9:49 AM ET at a price of $431.84. The fill model sets the limit price for sell orders when the high price is greater than or equal to the trigger price, fills the order when the ask close price is greater than or equal to the limit price, and sets the fill price to the maximum of the bid price and the limit price.

On July 2, 2021, the algorithm places a third LIT order to buy SPY with a trigger price of 431.70*andalimitpriceof*400. The fill model sets the fill price, but it doesn't fill the order because the limit price is too low. On the same day, the algorithm places a fourth LIT order to buy SPY with a trigger price of 400*andalimitpriceof*400. The fill model doesn't set the limit price for this order because SPY doesn't touch the trigger price before the backtest ends.

To reproduce these results, backtest the following algorithm:



Order Types

Stop Market Orders

Introduction

Stop market orders fill as a market order when the asset reaches a specific price. A buy stop market order triggers when the asset price is greater than or equal to the stop price. A sell stop market order triggers when the asset price is less than or equal to the stop price. You can use stop market orders to buy breakouts or to mitigate the risk of large losses. However, if the market gaps past your stop price, the order may fill at a worse price than the stop price you set. There is no guarantee that a stop market order fills at the stop price.

Place Orders

To send a stop market order, call the `StopMarketOrder` method and provide a `Symbol` , quantity, and stop price. You can also provide a tag and order properties to the `StopMarketOrder` method. If you do not have sufficient capital for the order, it is rejected.

```py
self.StopMarketOrder(symbol, quantity, stopPrice, tag, orderProperties)
```

To buy an asset with a stop market order, set the stop price above the current ask price.

```py
# When Bitcoin trades up to $60,000, buy 1 Bitcoin
self.StopMarketOrder("BTCUSD", 1, 60000)
```

To sell an asset with a stop market order, set the stop price below the current bid price.

```py
# When Bitcoin trades down to $43,000, sell 1 Bitcoin
self.StopMarketOrder("BTCUSD", -1, 43000)
```

Monitor Order Fills

Stop market orders fill as a market order when the security price passes the stop price. To monitor the fills of your order, save a reference to the order ticket .

```
# When XLK trades down to $130, sell 10 shares
ticket = self.StopMarketOrder("XLK", -10, 130)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, stop price, and tag of stop market orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
# Create a new order and save the order ticket
ticket = self.StopMarketOrder("SPY", -100, 415, tag="original tag")

# Update the order
update_settings = UpdateOrderFields()
update_settings.Quantity = -80
update_settings.StopPrice = 400
update_settings.Tag = "new tag"
response = ticket.Update(update_settings)

# Check if the update was successful
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateStopPrice`
- `UpdateTag`

```
response = ticket.UpdateQuantity(quantity, tag)
response = ticket.UpdateStopPrice(stopPrice, tag)
response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a stop market order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with stop market orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for stop market orders, see the **Orders** section of the brokerage model documentation .

Requirements

Stop market orders can be submitted at any time for all security types.

Example

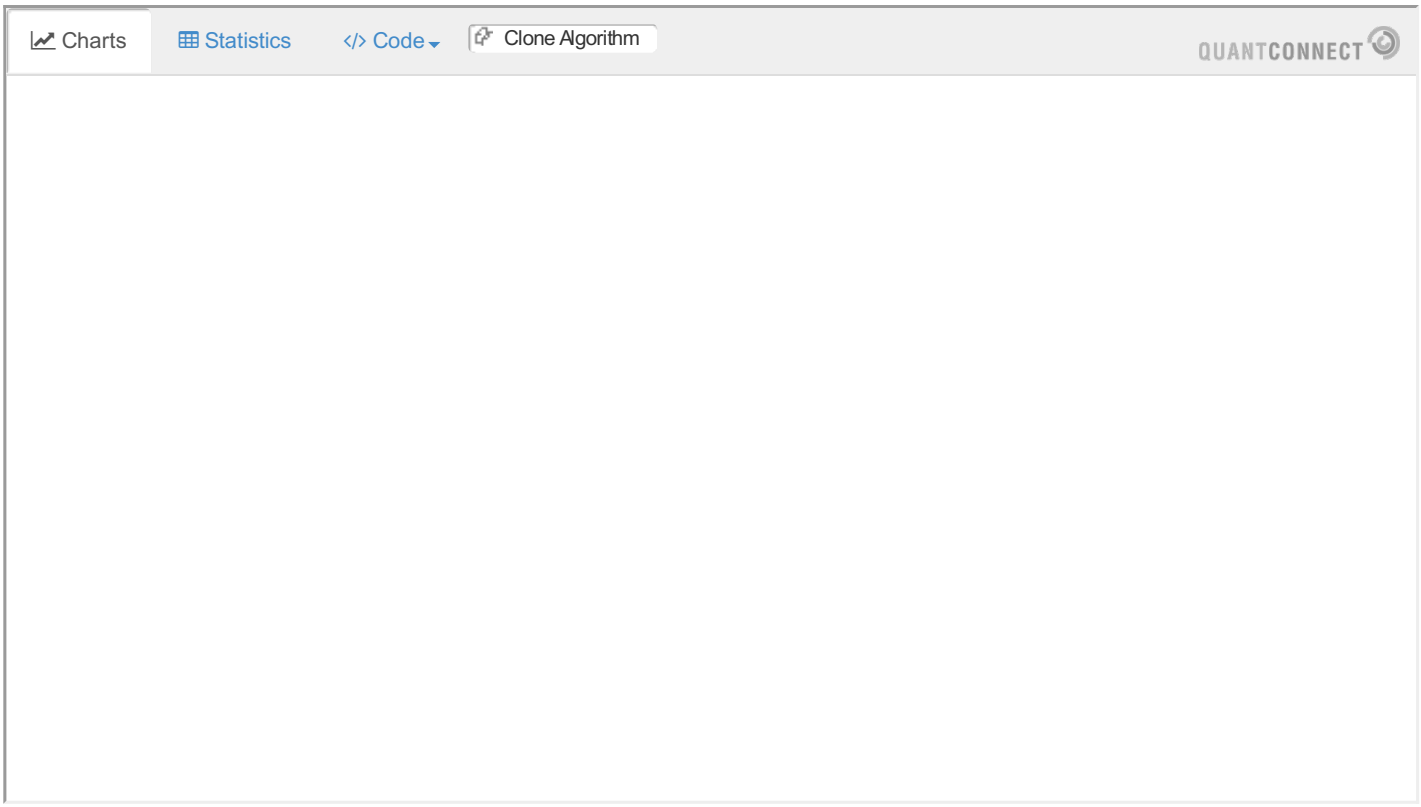
The following backtest verifies the `StopMarketOrder` behavior. On even days, the algorithm buys SPY at the current market price and sells when the price drops 1%. On odd days, the algorithm shorts SPY and buys when the price rises 1%. The following table shows the first four trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T13:31:00Z	SPY	429.10	-1	Market	Filled	-429.10	
2021-07-01T13:31:00Z	SPY	433.44	1	Stop Market	Filled	433.44	Stop Price: 433.43
2021-07-02T17:04:00Z	SPY	433.44	1	Market	Filled	433.44	
2021-07-02T17:04:00Z	SPY	429.00	-1	Stop Market	Filled	-429.00	Stop Price: 429.10

On July 1, 2021, the algorithm shorts SPY at 429.10*andthenbuysitbackat*433.44 when the stop price is 433.43.*Thestoppriceis*1429.10 is not the market price when the algorithm places the first two orders. 429.10*isthefillpriceatthebid*, *butit*'*snotfarfromthemarketpricebecause*429.1 □ 1.01 = 433.39.*Thefillpriceofthestopmarketorderis* 433.44, which, as expected, is higher than $433.43. The fill model assumes the worst-case scenario between the market price and the stop price. In this case, the worst-case scenario is the maximum of the market price and stop price.

On July 2, 2021, the algorithm buys SPY at 433.44*andsellsitat*429 when the stop price is 429.10.*Thestoppriceis*1433.44 is not the market price when the algorithm places the second two orders. 433.44*isthefillpriceattheask*, *butit*'*snotfarofffromthemarketpricebecause*433.44 □ 0.99 = 429.11.*Thefillpriceofthestopmarketorderis* 429, which, as expected, is lower than $429.10. The fill model assumes the worst-case scenario between the market price and the stop price. In this case, the worst-case scenario is the minimum of the market price and stop price.

To reproduce these results, backtest the following algorithm:


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Order Types

Stop Limit Orders

Introduction

Stop limit orders are instructions to place a limit order once an asset passes a specific price level. They are similar to stop market orders , but use limit orders instead of market orders to give you more control over the fill price. A buy stop limit order has a stop price above the current market price and a sell stop limit order has a stop price below the current market price. In effect, they are the opposite of limit if touched orders .

Place Orders

To send a stop limit order, call the `StopLimitOrder` method and provide a `Symbol` , quantity, stop price, and limit price. You can also provide a tag and order properties to the `StopLimitOrder` method. If you do not have sufficient capital for the order, it is rejected.

```py
self.StopLimitOrder(symbol, quantity, stop_rice, limit_price, tag, order_properties)
```

To buy an asset with a stop limit order that has a marketable limit price, set the stop price above the current market price and set the limit price above the stop price.

```py
# When Bitcoin trades up to $59,000, buy 1 Bitcoin with a limit order at $62,000
self.StopLimitOrder("BTCUSD", 1, 59000, 62000)
```

To buy an asset with a stop limit order that has an unmarketable limit price, set the stop price above the current market price and set the limit price below the stop price.

```py
# When Bitcoin trades up to $52,000, buy 1 Bitcoin with a limit order at $41,000
self.StopLimitOrder("BTCUSD", 1, 52000, 41000)
```

To sell an asset with a stop limit order that has an unmarketable limit price, set the stop price below the current market price and set the limit price above the stop price.

```py
# When Bitcoin trades down to $49,000, sell 1 Bitcoin with a limit order at $58,000
self.StopLimitOrder("BTCUSD", -1, 49000, 58000)
```

To sell an asset with a stop limit order that has a marketable limit price, set the stop price below the current market price and set the limit price below the stop price.

```
# When Bitcoin trades down to $37,000, sell 1 Bitcoin with a limit order at $34,000
self.StopLimitOrder("BTCUSD", -1, 37000, 34000)
```

Monitor Order Fills

Stop limit orders fill as limit orders when the security price passes the stop price. To monitor the fills of your order, save a reference to the order ticket .

```
# When GLD trades up to $200, buy 10 shares with a limit order at $205
ticket = self.StopLimitOrder("GLD", 10, 200, 205)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, stop price, limit price, and tag of stop limit orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request..

```
# Create a new order and save the order ticket
ticket = self.StopLimitOrder("SPY", -10, 400, 390, tag="original tag")

# Update the order
update_settings = UpdateOrderFields()
update_settings.Quantity = -15
update_settings.StopPrice = 415
update_settings.LimitPrice = 395
update_settings.Tag = "new tag"
response = ticket.Update(update_settings)

# Check if the update was successful
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateStopPrice`
- `UpdateTag`

```
PY
response = ticket.UpdateLimitPrice(limitPrice, tag)
response = ticket.UpdateQuantity(quantity, tag)
response = ticket.UpdateStopPrice(stopPrice, tag)
response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
PY
update_requests = ticket.UpdateRequests()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with stop limit orders, set the brokerage model to a brokerage that supports them.

```
PY
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for stop limit orders, see the **Orders** section of the brokerage model documentation .

Requirements

Stop limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Example

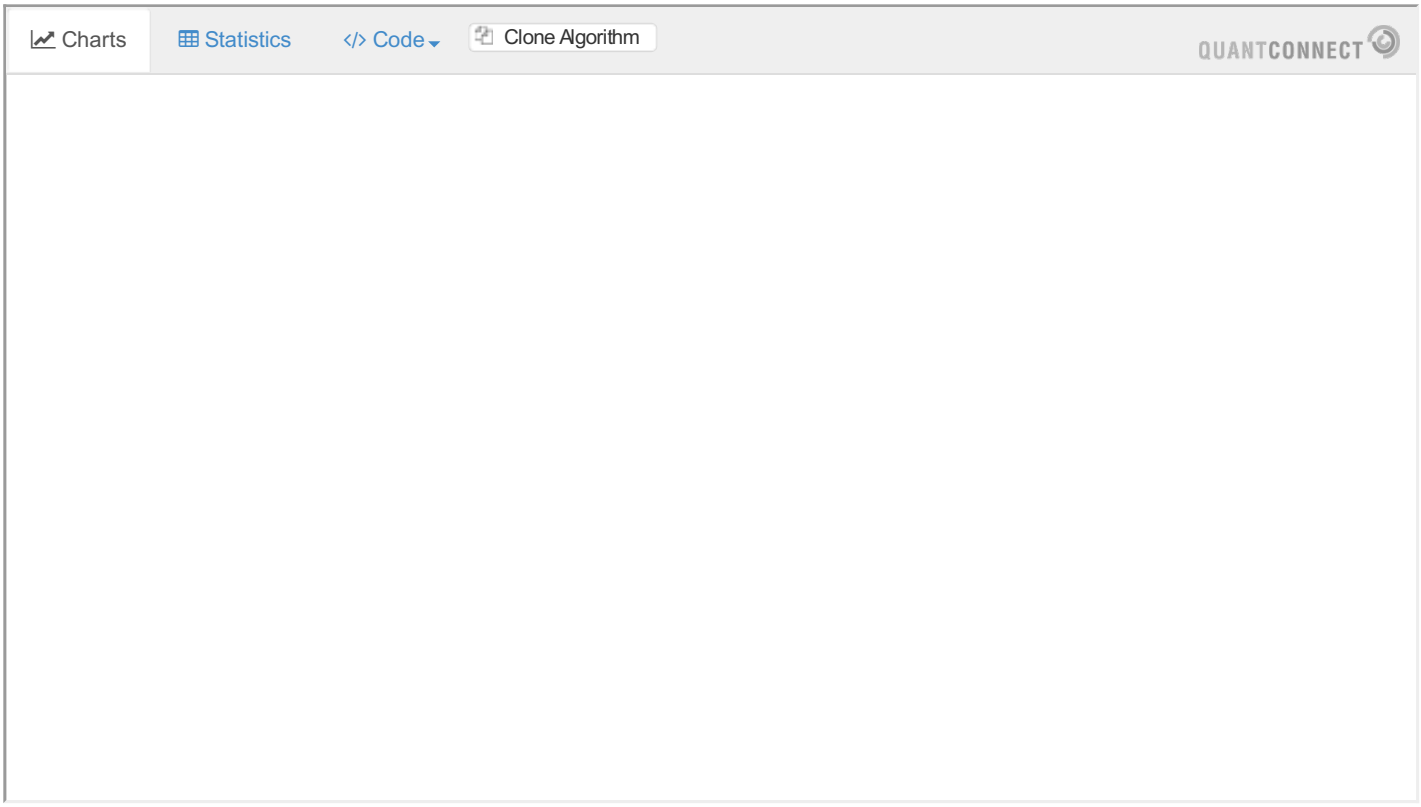
The following backtest verifies the `StopLimitOrder` behavior. The following table shows the trades in the backtest:

Submitted Time	Filled Time	Symbol	Stop Price	Limit Price	Filled Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	2021-07-01T09:37:00Z	SPY	428.95	429.00	429.00	-10	Stop Limit	Filled	-4290.00	Stop Price: ¤428.95 Limit Price: ¤429.00
2021-07-02T09:31:00Z	2021-07-02T10:12:00Z	SPY	431.80	431.75	431.75	10	Stop Limit	Filled	4317.50	Stop Price: ¤431.80 Limit Price: ¤431.75

On July 1, 2021 at 9:31 AM Eastern Time (ET), the algorithm places a stop limit order to sell SPY with a stop price of 428.95*andalimitpriceof*429. At the time of the order submission, the ask high price is 429.16*andtheaskclosepriceis*429.11. The order fills at 9:37 AM ET at a price of $429. The fill model triggers the stop for buy orders when the ask high price is greater than the stop price, fills the order when the ask close price is less than the limit price, and sets the fill price to the maximum of the ask high price and the limit price.

On July 2, 2021 at 9:31 AM ET, the algorithm places a stop limit order to buy SPY with a stop price of 431.80*andalimitpriceof*431.75. At the time of the order submissions, the bid low price is 431.49*andthebidclosepriceis* 431.54. The order fills at 10:12 AM ET at a price of $431.75. The fill model triggers the stop for sell orders when bid low price is less than the stop price, fills the order when the ask close price is less than the limit price, and sets the fill price to the minimum of the ask high price and the limit price.

To reproduce these results, backtest the following algorithm:



Order Types

Market On Open Orders

Introduction

Market on open (MOO) orders fill at the official opening auction price for a security. Market that operate 24/7 don't support MOO orders.

Place Orders

To send a MOO order, call the `MarketOnOpenOrder` method and provide a `Symbol` and quantity. Submit the order at least two minutes before the market opens to be included in the opening auction. If you do not have sufficient capital for the order, it's rejected.

```py
# Buy 100 shares of IBM at the market open
self.MarketOnOpenOrder("IBM", 100)

# Sell 100 shares of IBM at the market open
self.MarketOnOpenOrder("IBM", -100)
```

You can provide a tag and order properties to the `MarketOnOpenOrder` method.

```py
self.MarketOnOpenOrder(symbol, quantity, tag=tag, orderProperties=order_properties)
```

Monitor Order Fills

MOO orders fill at the opening auction. If the auction is heavily skewed to one side of the market, your MOO order may not fill. To monitor the fills of your order, save a reference to the order ticket .

```py
# Buy 10 shares of SPY at the market open
ticket = self.MarketOnOpenOrder("SPY", 10)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

You won't know the fill price of the order until after the market opens. If the asset price moves before the market open to where you can't afford the quantity of the order, the brokerage rejects your order. To increase the probability of a successful trade, leave a sufficient buying power buffer to handle daily price gaps.

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity and tag of MOO orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success

or failure of the update request.

```
# Create a new order and save the order ticket
ticket = self.MarketOnOpenOrder("AAPL", 100, "original tag")

# Update the order
update_settings = UpdateOrderFields()
update_settings.Quantity = 75
update_settings.Tag = "new tag"
response = ticket.Update(update_settings)

# Check if the update was successful
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateTag`

```
response = ticket.UpdateQuantity(quantity, tag)

response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a MOO order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with MOO orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for MOO orders, see the **Orders** section of the brokerage model documentation .

Requirements

You can submit MOO orders at any time for Equity, Equity Options, and Index Options.

Markets that operate 24/7 don't support MOO orders. The Forex market doesn't operate during the weekend. However, MOO orders are invalid if you place them after this market closes for the weekend.

MOO orders don't support the `GoodTilDate` time in force . If you submit a MOO order with the `GoodTilDate` time in force, LEAN automatically adjusts the time in force to be `GoodTilCanceled` .

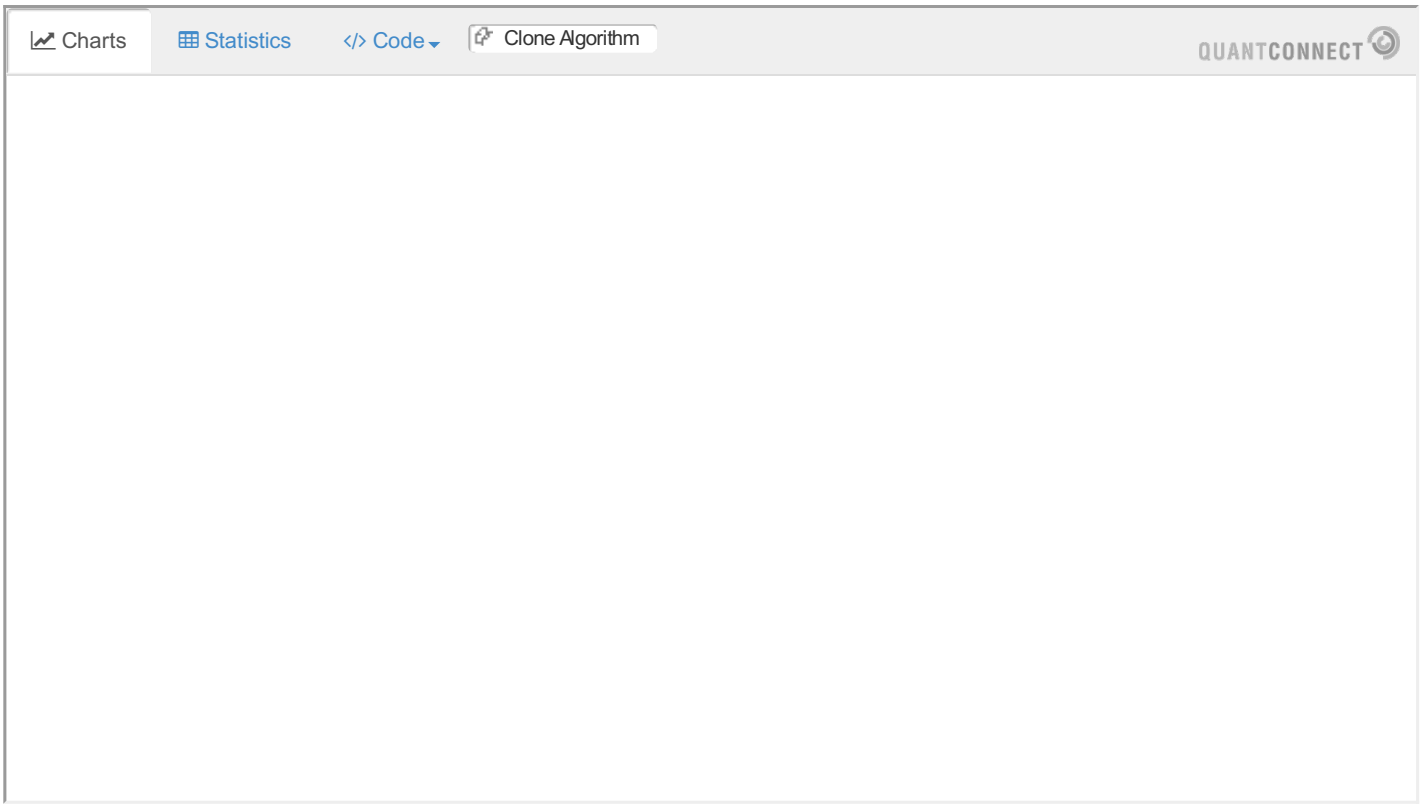
Example

The following backtest verifies the `MarketOnOpenOrder` behavior. The following table shows the first trade in the backtest:

Submitted Time	Filled Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T10:31:00Z	2021-07-02T09:31:00Z	SPY	431.67	10	Market On Open	Filled	4316.70	

On July 1, 2021 at 10:31 AM Eastern Time (ET), the algorithm places a market on open order to buy SPY. The fill model fills the order on July 2, 2021 at 9:31 AM at a price of $431.67, which is the official opening auction price for July 2, 2021.

To reproduce these results, backtest the following algorithm:


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Order Types

Market On Close Orders

Introduction

Market on close (MOC) orders fill at the official closing auction price for a security. Markets that operate 24/7 don't support MOC orders.

Place Orders

To send a MOC order, call the `MarketOnCloseOrder` method with a `Symbol` and quantity. If you don't have sufficient capital for the order, it is rejected.

```
PY
# Buy 100 shares of AAPL at the market open
self.MarketOnCloseOrder("AAPL", 100)

# Sell 100 shares of AAPL at the market open
self.MarketOnCloseOrder("AAPL", -100)
```

You can provide a tag and order properties to the `MarketOnCloseOrder` method.

```
PY
self.MarketOnCloseOrder(symbol, quantity, tag=tag, orderProperties=order_properties)
```

By default, you must place MOC orders at least 15.5 minutes before the close, but some exchanges let you submit them closer to the market closing time. To adjust the buffer period that's required, set the `MarketOnCloseOrder.SubmissionTimeBuffer` property.

```
PY
MarketOnCloseOrder.SubmissionTimeBuffer = timedelta(minutes=10)
```

You can also place MOC orders after the market close.

Monitor Order Fills

MOC orders fill at the closing auction. If the auction is heavily skewed to one side of the market, your MOC order may not fill. To monitor the fills of your order, save a reference to the order ticket .

```
PY
# Buy 10 shares of SPY at the market close
ticket = self.MarketOnCloseOrder("SPY", 10)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

You won't know the fill price of the order until after the market closes. If the asset price moves before the market close to where you can't afford the quantity of the order, the brokerage rejects your order. To increase the probability of a

successful trade, leave a sufficient buying power buffer to handle the price movements before the close.

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity and tag of MOC orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
PY
# Create a new order and save the order ticket
ticket = self.MarketOnOpenOrder("SLV", 25, "original tag")

# Update the order
update_settings = UpdateOrderFields()
update_settings.Quantity = 50
update_settings.Tag = "new tag"
response = ticket.Update(update_settings)

# Check if the update was successful
if response.IsSuccess:
    self.Debug("Order updated successfully")
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateTag`

```
PY
response = ticket.UpdateQuantity(quantity, tag)

response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
PY
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a MOC order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
PY
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

PY

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with MOC orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

PY

To check if your brokerage has any special requirements for MOC orders, see the **Orders** section of the brokerage model documentation .

Requirements

By default, you must place MOC orders at least 15.5 minutes before the close, but some exchanges let you submit them closer to the market closing time. To adjust the buffer period that's required, set the `MarketOnCloseOrder.SubmissionTimeBuffer` property.

```
MarketOnCloseOrder.SubmissionTimeBuffer = timedelta(minutes=10)
```

PY

Markets that operate 24/7 don't support MOC orders. The Forex market doesn't operate during the weekend.

MOC orders don't support the `GoodTilDate` time in force . If you submit a MOC order with the `GoodTilDate` time in force, LEAN automatically adjusts the time in force to be `GoodTilCanceled` .

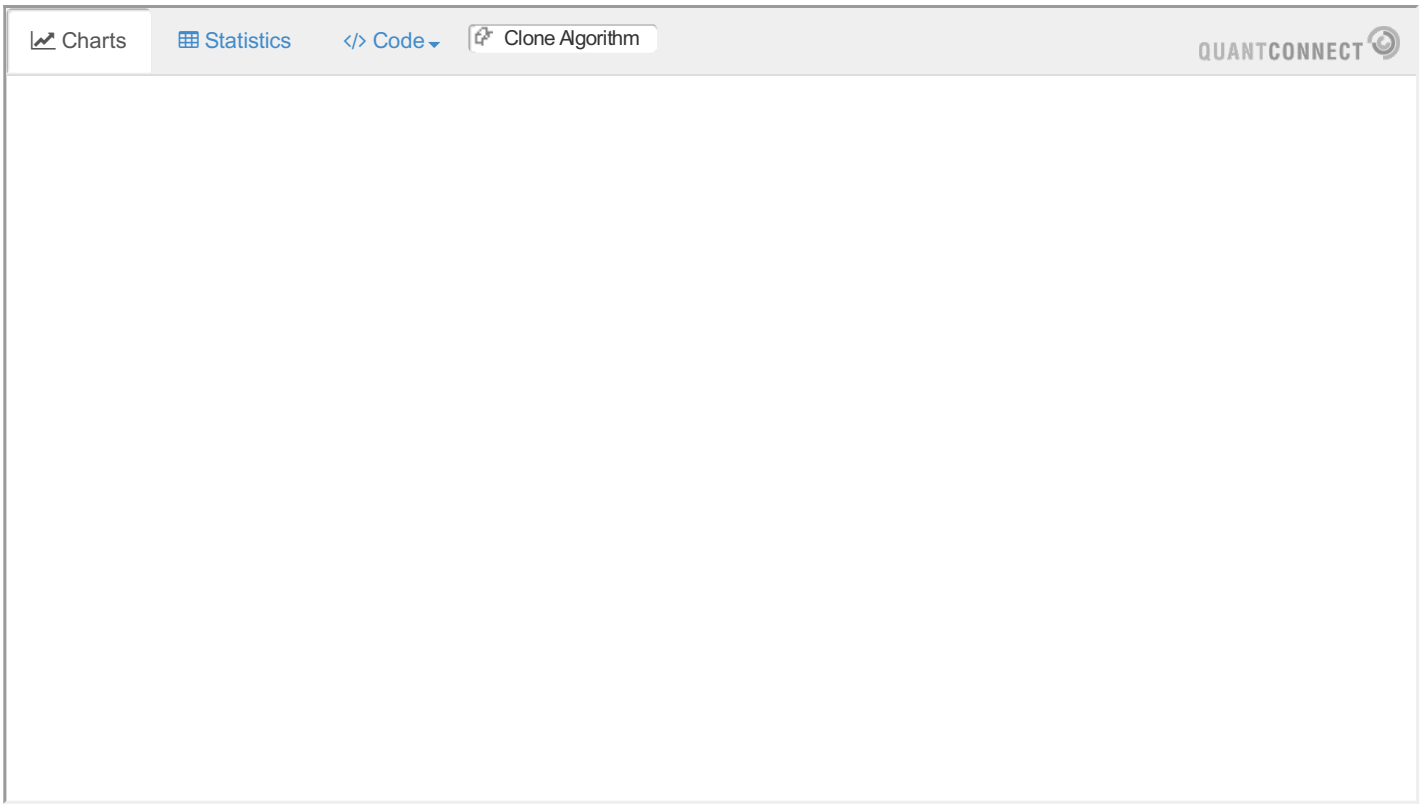
Example

The following backtest verifies the `MarketOnCloseOrder` behavior. The following table shows the first trade in the backtest:

Submitted Time	Filled Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T10:31:00Z	2021-07-01T16:00:00Z	SPY	430.43	10	Market On Close	Filled	4304.30	

On July 1, 2021 at 10:31 AM Eastern Time (ET), the algorithm places a market on open order to buy SPY. The fill model fills the order on July 1, 2021 at 4:00 PM ET at a price of $430.43, which is the official closing auction price for July 1, 2021.

To reproduce these results, backtest the following algorithm:


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Order Types

Combo Market Orders

Introduction

Combo market orders are individual orders that contain market orders for muliple securities. Combo market orders currently only work for trading Option contracts.

Place Orders

To send a combo market order, create multiple `Leg` objects to represent the legs of the combo order, then call the `ComboMarketOrder` method. At least one leg must have a positive quantity and at least one leg must have a negative quantity. The legs must each target a unique contract. If you don't have sufficient capital for the order, it's rejected. By default, combo market orders are synchronous and fill immediately.

```
PY
for canonical_symbol, chain in slice.OptionChains.items():
    # Select contracts
    contracts = [c for c in chain][:2]
    if len(contracts) < 2:
        return

    # Create order legs
    legs = []
    quantities = [1, -1]
    for i, contract in enumerate(contracts):
        legs.append(Leg.Create(contract.Symbol, quantities[i]))

    # Place order
    self.ComboMarketOrder(legs, 1)
```

The quantity of the legs sets the ratio of the leg orders while the quantity argument of the `ComboMarketOrder` method sets the combo order size and acts as a global multiplier. In the preceding example, if we set the global multiplier to two, then the algorithm buys two units of the first contract and sells two units of the second contract. The quantity argument of the `ComboMarketOrder` method also sets the order direction of the combo order, which affects how the fill model fills the order.

You can also provide a tag and order properties to the `ComboMarketOrder` method.

```
PY
self.ComboMarketOrder(legs, quantity, tag=tag, orderProperties=order_properties)
```

Monitor Order Fills

If the brokerage has sufficient liquidity in their order book, combo market orders fill immediately. Otherwise, you get partial fills. To monitor the fills of your order, save a reference to the order tickets .

```
ticket = self.ComboMarketOrder(legs, 1)
for ticket in tickets:
    self.Debug(f"Symbol: {ticket.Symbol}; Quantity filled: {ticket.QuantityFilled}; Fill price:
{ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Synchronous Timeouts

Combo market orders are synchronous by default, so your algorithm waits for the order to fill before moving to the next line of code. If your order takes longer than five seconds to fill, your algorithm continues executing even if the trade isn't filled. To adjust the timeout period, set the `Transactions.MarketOrderFillTimeout` property.

```
 # Adjust the market fill-timeout to 30 seconds.
self.Transactions.MarketOrderFillTimeout = timedelta(seconds=30)
```

Place Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders, set the `asynchronous` argument to true.

```
self.ComboMarketOrder(legs, quantity, True)
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with combo market orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for combo market orders, see the **Orders** section of the brokerage model documentation .

Requirements

Combo market orders must be submitted during market hours for all security types.

If your algorithm place combo market orders at or after the last minute of regular market hours, they will be converted into market-on-open orders and will have to observe their for the following asset types:

1. Equities
2. Equity Options
3. Forex
4. CFDs
5. Index Options

Combo market orders for Futures and Future Options can be submitted during extended market hours, or they will be invalid.

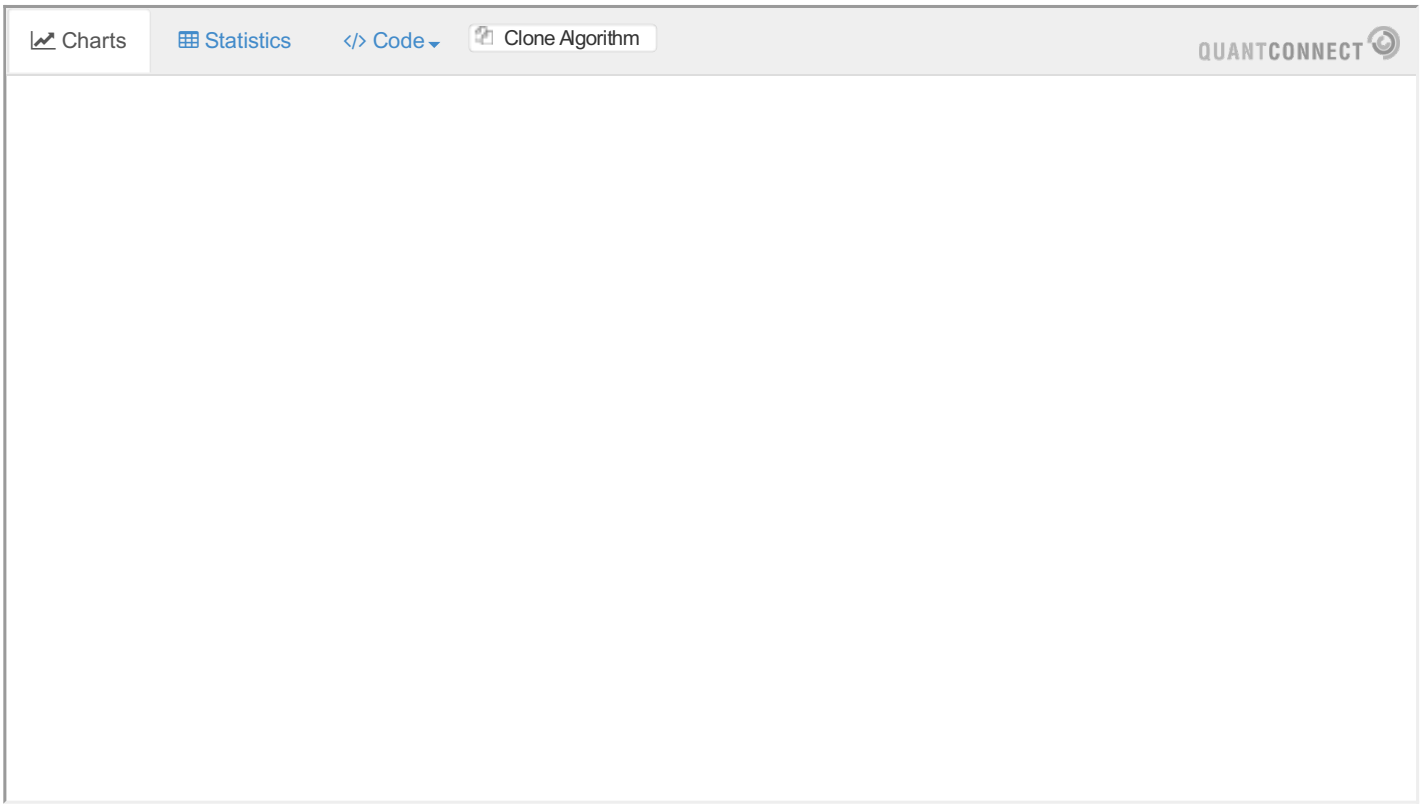
Example

The following backtest verifies the `ComboMarketOrder` behavior. The algorithm buys one contract and sells one contract at the same time. The following table shows the two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2015-12-24T09:31:00Z	GOOG 16011SC00745000	16.90	1	Buy	Filled	16.90	
2015-12-24T09:31:00Z	GOOG 160115C00747500	14.20	-1	Sell	Filled	-14.20	

On December 24, 2015, the algorithm buys GOOG 16011SC00745000 at 16.90*and*sells*GOOG*160115*C*00747500*at*14.20. The fill model fills the buy order at the ask close price and fills the sell order at the bid close price.

To reproduce these results, backtest the following algorithm:



Order Types

Combo Limit Orders

Introduction

Combo limit orders are individual orders that contain limit orders for muliple securities. Combo limit orders are different from combo leg limit orders because you must set the limit price of all the leg orders to be the same with combo limit orders. With combo leg limit orders, you can create the order without forcing each leg to have the same limit price. Combo limit orders currently only work for trading Option contracts and their underlying Equities.

Place Orders

To send a combo limit order, create multiple `Leg` objects to represent the legs of the combo order, then call the `ComboLimitOrder` method. At least one leg must have a positive quantity and a tleast one leg must have a negative quantity. The legs must each target a unique contract, but don't set the `OrderPrice` property for any of the legs. If you don't have sufficient capital for the order, it's rejected.

```py
for canonical_symbol, chain in slice.OptionChains.items():
    # Select contracts
    contracts = [c for c in chain][:2]
    if len(contracts) < 2:
        return

    # Create order legs
    legs = []
    quantities = [1, -1]
    for i, contract in enumerate(contracts):
        legs.append(Leg.Create(contract.Symbol, quantities[i]))

    # Calculate limit price
    limit_price = round(sum([self.Securities[leg.Symbol].Close for leg in legs]) * 0.95, 2)

    # Place order
    self.ComboLimitOrder(legs, 1, limit_price)
```

The quantity of the legs sets the ratio of the leg orders while the quantity argument of the `ComboLimitOrder` method sets the combo order size and acts as a global multiplier. In the preceding example, if we set the global multiplier to two, then the algorithm buys two units of the first contract and sells two units of the second contract. The quantity also sets the order direction of the combo limit order, which affects how the fill model fills the order.

You can also provide a tag and order properties to the `ComboLimitOrder` method.

```py
self.ComboLimitOrder(legs, quantity, limit_price, tag=tag, orderProperties=order_properties)
```

Monitor Order Fills

Combo limit orders fill all the legs at the same time. Each leg can fill when the sum of the security prices of the legs pass the limit price of the combo order. To monitor the fills of your order, save a reference to the order tickets .

```
tickets = self.ComboLimitOrder(legs, 1, limit_price)
for ticket in tickets:
    self.Debug(f"Symbol: {ticket.Symbol}; Quantity filled: {ticket.QuantityFilled}; Fill price:
{ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, limit price, and tag of the limit orders in each leg until the combo order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . To update the limit price of the combo order, you only need to update the limit price of one of the leg orders. The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
# Create a new order and save the order tickets
tickets = self.ComboLimitOrder(legs, 1, limit_price)

# Update the leg orders
for ticket in tickets:
    update_settings = UpdateOrderFields()
    update_settings.Quantity = 2 * np.sign(ticket.Quantity)
    update_settings.LimitPrice = ticket.Get(OrderField.LimitPrice) + 0.01
    update_settings.Tag = f"Update #{len(ticket.UpdateRequests) + 1}"
    response = ticket.Update(update_settings)

    # Check if the update was successful
    if response.IsSuccess:
        self.Debug(f"Order updated successfully for {ticket.Symbol}")
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTag`

```
response = ticket.UpdateLimitPrice(limitPrice, tag)

response = ticket.UpdateQuantity(quantity, tag)

response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a combo limit order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it

from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with combo limit orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for combo limit orders, see the **Orders** section of the brokerage model documentation .

Requirements

Combo limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

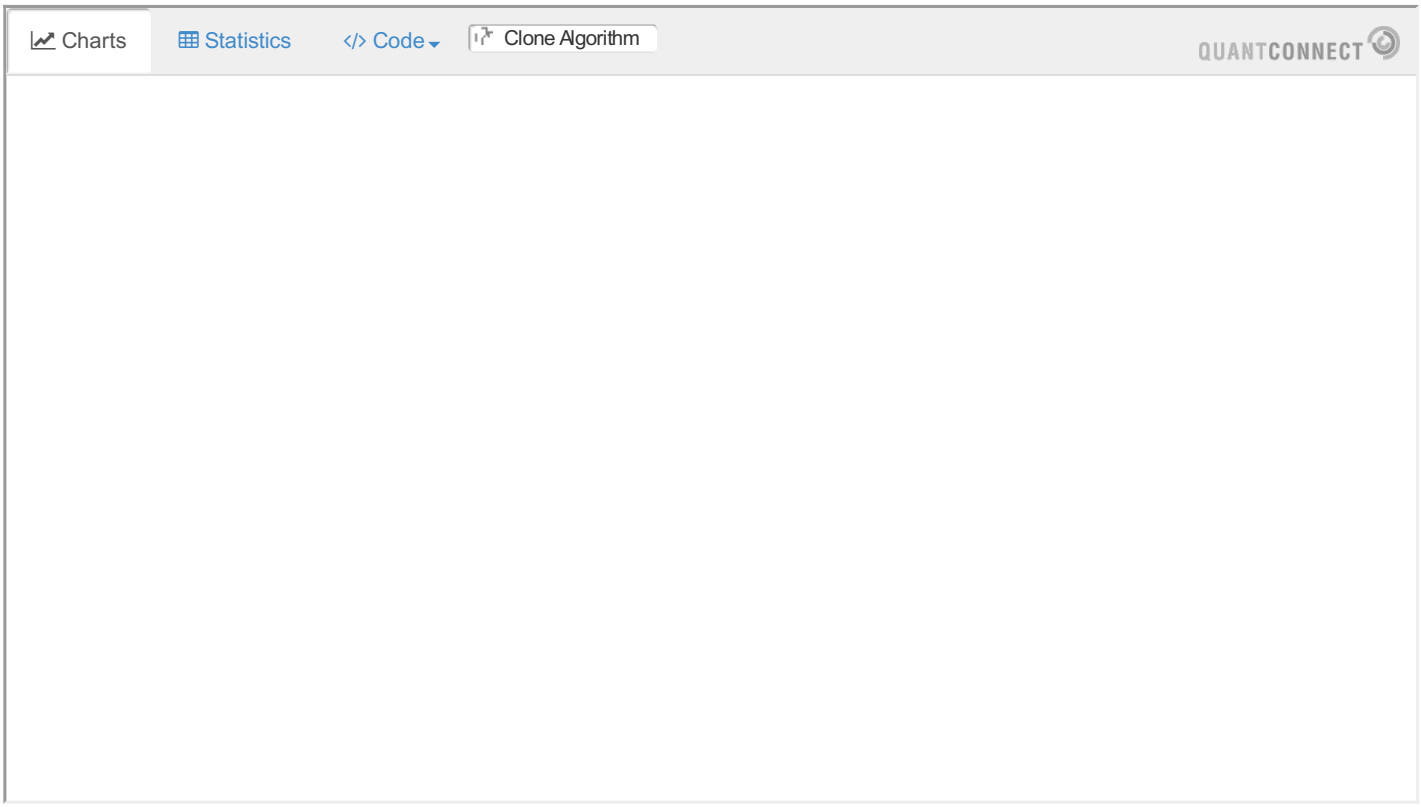
Example

The following backtest verifies the `ComboLimitOrder` behavior. The algorithm buys one contract and sells one contract at the same time. The following table shows the two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2015-12-24T09:31:00Z	GOOG 16011SC00745000	16.50	1	Buy	Filled	16.50	
2015-12-24T09:31:00Z	GOOG 160115C00747500	14.60	-1	Sell	Filled	-14.60	

On December 24, 2015 at 9:31 AM Eastern Time (ET), the algorithm places a combo limit order to buy one GOOG 16011SC00745000 contract and sell one GOOG 160115C00747500 contracts. The limit price is 75% of the contract spread, which equals 2.02. *The combo order doesn't fill immediately because the contract spread* 2.40 > 2.02. *By* 9:36 *AM*, *the spread drops to* 1.90, which is below the limit price, so the fill model fills the combo limit order.

To reproduce these results, backtest the following algorithm:



Order Types

Combo Leg Limit Orders

Introduction

Combo leg limit orders are individual orders that contain limit orders for muliple securities. Combo leg limit orders are different from combo limit orders because you can create combo leg limit orders without forcing each leg to have the same limit price. Combo leg limit orders currently only work for trading Option contracts.

Place Orders

To send a combo leg limit order, create multiple `Leg` objects to represent the legs of the combo order, then call the `ComboLegLimitOrder` method. The legs must each target a unique contract. At least one leg must have a positive quantity and at least one leg must have a negative quantity. If you don't have sufficient capital for the order, it's rejected.

```py
for canonical_symbol, chain in slice.OptionChains.items():
    # Select contracts
    contracts = [c for c in chain][:2]
    if len(contracts) < 2:
        return

    # Create order legs
    legs = []
    quantities = [1, -1]
    factors = [0.98, 1.02]
    for i, contract in enumerate(contracts):
        legs.append(Leg.Create(contract.Symbol, quantities[i], contract.LastPrice * factors[i]))

    # Place order
    self.ComboLegLimitOrder(legs, 1)
```

The quantity of the legs sets the ratio of the leg orders while the quantity argument of the `ComboLegLimitOrder` method sets the combo order size and acts as a global multiplier. In the preceding example, if we set the global multiplier to two, then the algorithm buys two units of the first contract and sells two units of the second contract. The quantity also sets the order direction of the combo limit order, which affects how the fill model fills the order.

You can also provide a tag and order properties to the `ComboLegLimitOrder` method.

```py
self.ComboLegLimitOrder(legs, quantity, tag=tag, orderProperties=order_properties)
```

Monitor Order Fills

Combo leg limit orders fill all the legs at the same time. Each leg can fill when the security price passes the limit price of the leg. To monitor the fills of your order, save a reference to the order tickets .

```
tickets = self.ComboLegLimitOrder(legs, 1)
for ticket in tickets:
    self.Debug(f"Symbol: {ticket.Symbol}; Quantity filled: {ticket.QuantityFilled}; Fill price:
{ticket.AverageFillPrice}")
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, limit price, and tag of the leg limit orders until the combo order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
# Create a new order and save the order tickets
tickets = self.ComboLegLimitOrder(legs, 1)

# Update the leg orders
for ticket in tickets:
    direction = np.sign(ticket.Quantity)
    update_settings = UpdateOrderFields()
    update_settings.Quantity = 2 * direction
    update_settings.LimitPrice = ticket.Get(OrderField.LimitPrice) + 0.01 * direction
    update_settings.Tag = f"Update #{len(ticket.UpdateRequests) + 1}"
    response = ticket.Update(update_settings)

    # Check if the update was successful
    if response.IsSuccess:
        self.Debug(f"Order updated successfully for {ticket.Symbol}")
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTag`

```
response = ticket.UpdateLimitPrice(limitPrice, tag)

response = ticket.UpdateQuantity(quantity, tag)

response = ticket.UpdateTag(tag)
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
update_requests = ticket.UpdateRequests()
```

Cancel Orders

To cancel a combo leg limit order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it

from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
response = ticket.Cancel("Cancelled Trade")
if response.IsSuccess:
    self.Debug("Order successfully cancelled")
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `None` if the order hasn't been cancelled.

```
request = ticket.CancelOrderRequest()
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with combo leg limit orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for combo leg limit orders, see the **Orders** section of the brokerage model documentation .

Requirements

Combo leg limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Example

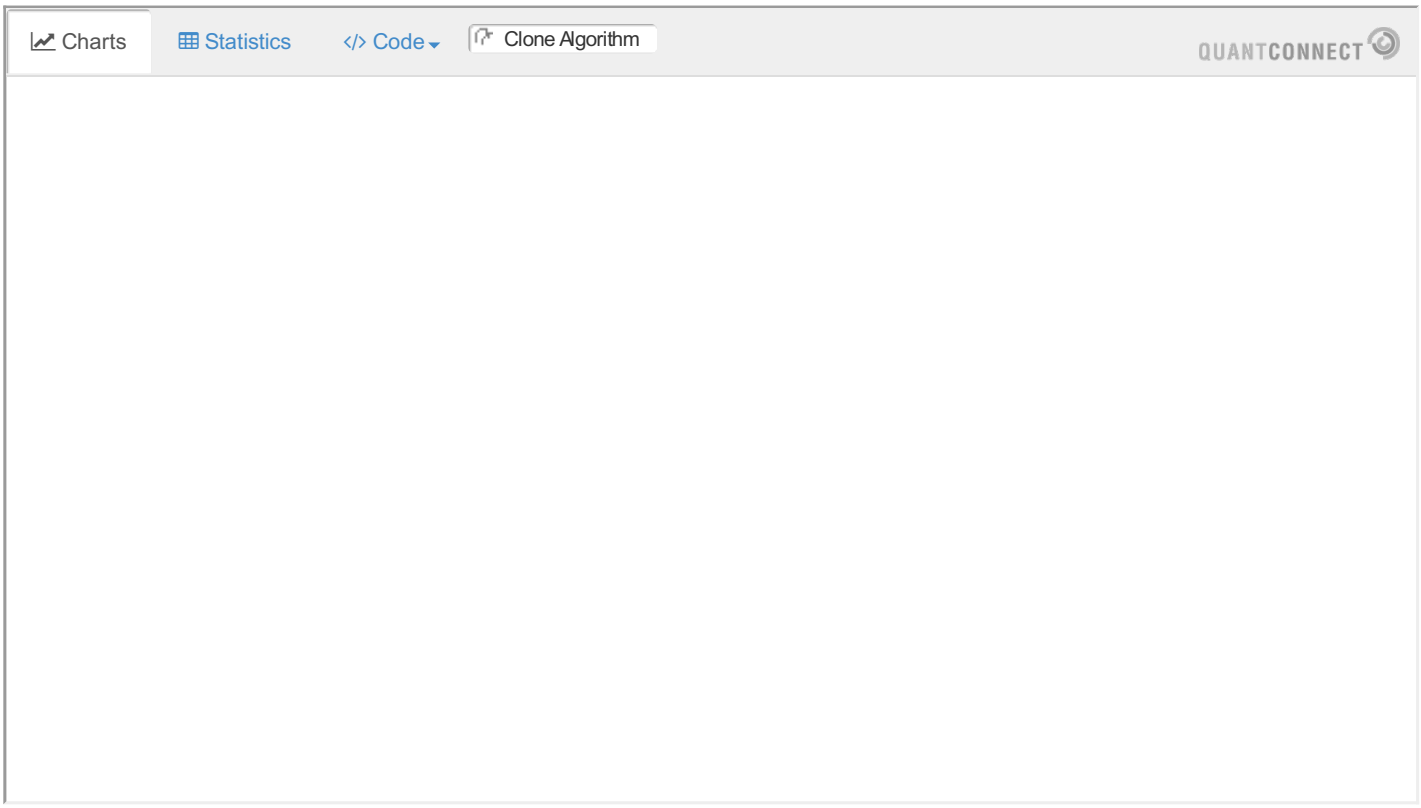
The following backtest verifies the `ComboLegLimitOrder` behavior. The algorithm buys one contract and sells one contract at the same time. The following table shows the two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2015-12-24T09:31:00Z	GOOG 16011SC00745000	16.10	2	Buy	Filled	32.20	Update #72
2015-12-24T09:31:00Z	GOOG 160115C00747500	14.11515	-2	Sell	Filled	-28.2303	Update #72

On December 24, 2015 at 9:31 AM Eastern Time (ET), the algorithm places a combo leg limit order to buy one GOOG 16011SC00745000 contract and sell two GOOG 160115C00747500 contracts. The limit price of both orders is 99.9% of the respective contract price, which is 16.2837*forGOOG*16011*SC*00745000*and*14.83515 for GOOG 160115C00747500. The combo order doesn't fill immediately, so the algorithm updates the leg orders at each time step. During the first

update, the algorithm sets the quantity of the GOOG 160115C00747500 leg to -2. During each update, the limit price moves 0.01 *closer to the market*. *That is*, *the limit price of GOOG* 16011 *SC* 00745000 *increases by* 0.01 and the limit price of GOOG 160115C00747500 decreases by $0.01. After the 72nd update, the ask low price is below the limit price of the leg to buy GOOG 16011SC00745000 and the bid high price is above the limit price of the leg to sell GOOG 160115C00747500, so the fill model fills the combo leg limit order at 10:44 AM ET.

To reproduce these results, backtest the following algorithm:



Order Types

Option Exercise Orders

Introduction

If you buy an Option contract, you can exercise your right to buy or sell the underlying asset at the strike price. However, you don't need to exercise it. You can sell the Option or let it expire worthless if it's out of the money. If you hold a long position in an Option that expires in the money, LEAN automatically exercises it at the expiration date. If you sell an Option contract and the buyer exercises their right to buy or sell the underlying asset, you are assigned the Option and must trade with the buyer at the strike price.

Place Orders

You can exercise American-style Option contracts anytime before they expire. Depending on your brokerage and the delivery method, you may be able to exercise European-style Option contracts on their expiration date. To exercise an Option, call the `ExerciseOption` method with the Option contract `Symbol` and a quantity. If you do not have sufficient capital for the order, it's rejected. By default, Option exercise orders are synchronous and fill immediately.

```
PY
ticket = self.ExerciseOption(contract_symbol, quantity)
```

You can provide a tag and order properties to the `ExerciseOption` method.

```
PY
self.ExerciseOption(symbol, quantity, tag=tag, orderProperties=order_properties)
```

Monitor Order Fills

To monitor the fills of your order, save a reference to the order ticket .

```
PY
ticket = self.ExerciseOption(contract_symbol, quantity)
self.Debug(f"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}")
```

For more information about how LEAN models Option exercise orders in backtests, see the Exercise Option model .

Synchronous Timeouts

Option exercise orders are synchronous by default, so your algorithm waits for the order to fill before moving to the next line of code. If your order takes longer than five seconds to fill, your algorithm continues executing even if the trade isn't filled. To adjust the timeout period, set the `Transactions.MarketOrderFillTimeout` property.

```
PY
 # Adjust the market fill-timeout to 30 seconds.
self.Transactions.MarketOrderFillTimeout = timedelta(seconds=30)
```

Place Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders, set the `asynchronous` argument to `True` .

```
ticket = self.ExerciseOption(contract_symbol, quantity, True)
```

Option Assignments

If you sell an Option in a backtest, LEAN can simulate an Option exercise order on behalf of the buyer. By default, LEAN scans your portfolio every hour. It considers exercising American-style Options if they are within 4 days of their expiration and it considers exercising European-style Options on their day of expiration. If you have sold an Option that's 5% in-the-money and the Option exercise order is profitable after the cost of fees, LEAN exercises the Option. For more information about how we simulate Option assignments, see the Assignment reality model.

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with Option exercise orders, set the brokerage model to a brokerage that supports them.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage)
```

To check if your brokerage has any special requirements for Option exercise orders, see the **Orders** section of the brokerage model documentation .

Requirements

Option exercise orders can only be submitted for option contracts with a long position. European-style options cannot be exercised before their expiration date.

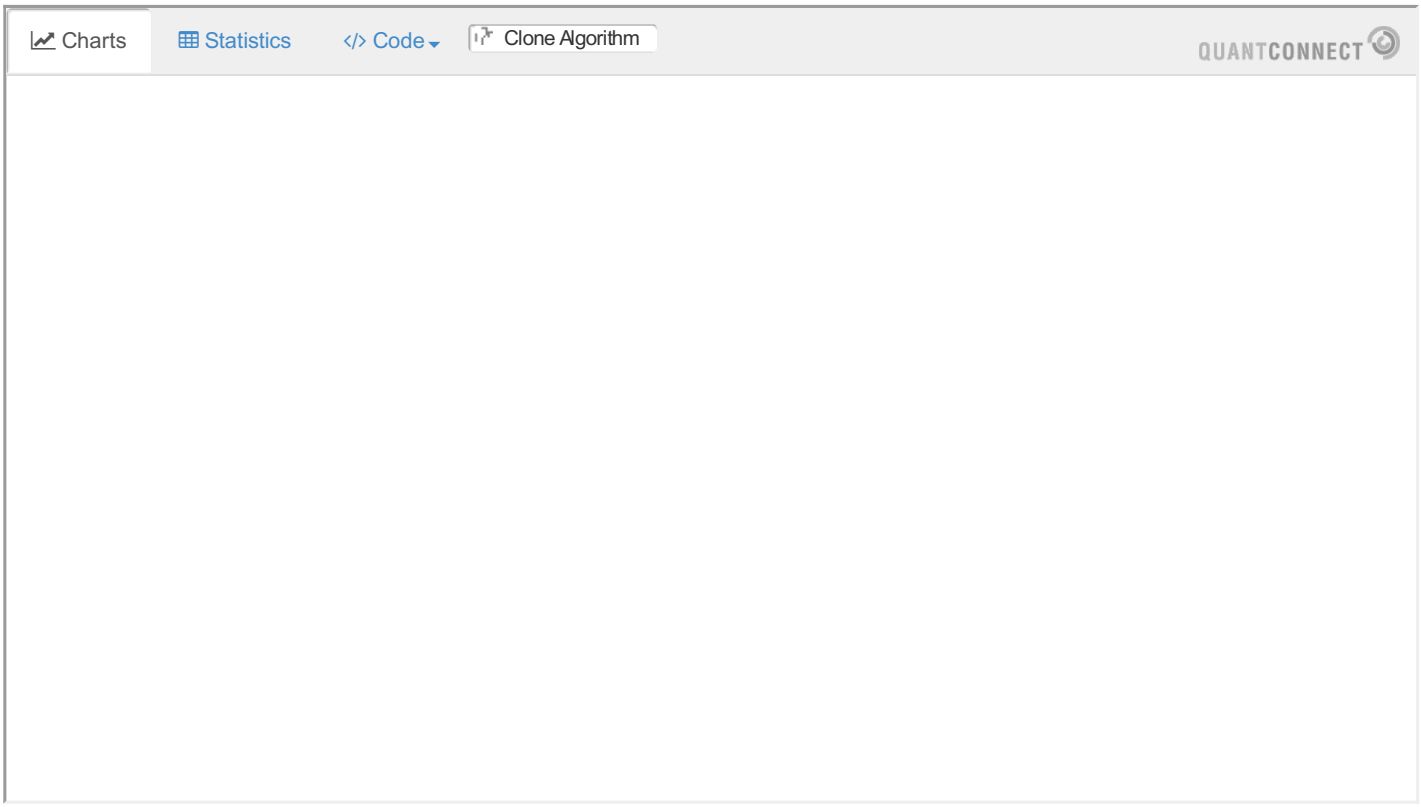
Example

The following backtest verifies the `ExerciseOption` behavior. The following table shows the first three trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	SPY 221216C00385000	345.30	1	Buy Market	Filled	34530.00	
2021-07-01T09:31:00Z	SPY 221216C00385000	0.00	-1	Sell Option Exercise	Filled	0.00	Automatic Exercise
2021-07-01T09:31:00Z	SPY	85.00	100	Sell Option Exercise	Filled	8500.00	Option Exercise

The algorithm first brought a deep ITM option based on the set option selection conditions by 345.30, *then exercise it actively*, *so* 100 *shares of SPY* (*in which* 1 *contract represents*) *was brought at its strike price at* 85.00, with the option discarded from the portfolio.

To reproduce these results, backtest the following algorithm:



Order Types

Other Order Types

Introduction

We are often asked to support other order types like one cancels the other, trailing stop, and multi-leg orders. Currently, LEAN doesn't support these order types, but we will add them over time. Part of the difficulty of implementing them is the incomplete brokerage support.

One Cancels the Other Orders

One cancels the other (OCO) orders are a set of orders that when one fills, it cancels the rest of the orders in the set. An example is to set a take-profit and a stop-loss order right after you enter a position. In this example, when either the take-profit or stop-loss order fills, you cancel the other order. OCO orders usually create an upper and lower bound on the exit price of a trade.

When you place OCO orders, their price levels are usually relative to the fill price of an entry trade. If your entry trade is a synchronous market order, you can immediately get the fill price from the order ticket. If your entry trade doesn't execute immediately, you can get the fill price in the OnOrderEvents event handler. Once you have the entry fill price, you can calculate the price levels for the OCO orders.

```py
# Get the fill price from the order ticket of a sync market order
self.ticket = self.MarketOrder("SPY", 1)
fill_price = self.ticket.AverageFillPrice

# Get the fill price from the OnOrderEvent event handler
def OnOrderEvent(self, orderEvent: OrderEvent) -> None:
    if orderEvent.Status == OrderStatus.Filled and self.ticket.OrderId == orderEvent.OrderId:
        fill_price = orderEvent.FillPrice
```

After you have the target price levels, to implement the OCO orders, you can place active orders or track the security price to simulate the orders.

Place Active Orders

To place active orders for the OCO orders, use a combination of limit orders and stop limit orders . Place these orders so that their price levels that are far enough apart from each other. If their price levels are too close, several of the orders can fill in a single time step. When one of the orders fills, in the `OnOrderEvent` event handler, cancel the other orders in the OCO order set.

```
self.stop_loss = None
self.take_profit = None

def OnOrderEvent(self, orderEvent: OrderEvent) -> None:
    if orderEvent.Status == OrderStatus.Filled:
        if orderEvent.OrderId == self.ticket.OrderId:
            self.stop_loss = self.StopMarketOrder(orderEvent.Symbol, -orderEvent.FillQuantity,
orderEvent.FillPrice*0.95)
            self.take_profit = self.LimitOrder(orderEvent.Symbol, -orderEvent.FillQuantity,
orderEvent.FillPrice*1.10)

        elif self.stop_loss is not None and orderEvent.OrderId == self.stop_loss.OrderId:
            self.take_profit.Cancel()

        elif self.take_profit is not None and orderEvent.OrderId == self.take_profit.OrderId:
            self.stop_loss.Cancel()
```

Simulate Orders

To simulate OCO orders, track the asset price in the `OnData` method and place market or limit orders when asset price reaches the take-profit or stop-loss level. The benefit of manually simulating the OCO orders is that both of the orders can't fill in the same time step.

```
def OnData(self, slice: Slice) -> None:
    if not self.Portfolio.Invested:
        ticket = self.MarketOrder("SPY", 1)
        self.entry_price = ticket.AverageFillPrice

    if slice.Bars.ContainsKey("SPY"):
        if slice.Bars["SPY"].Price >= self.entry_price * 1.10:
            self.Liquidate("SPY", -1, "take profit")

        elif slice.Bars["SPY"].Price <= self.entry_price * 0.95:
            self.Liquidate("SPY", -1, "stop loss")
```

Trailing Stop Loss Orders

A trailing stop loss order is a dynamic stop loss that moves towards the asset price as the asset trades away from the stop price. For example, if you set a trailing stop loss to 10%, the furthest away it will ever be from the asset price is 10%. If it's a sell trailing stop loss order that's below the asset price and the asset price increases such that the trailing stop is more than 10% away, the trailing stop loss moves up to be 10% below the asset price. If the asset price decreases, the trailing stop loss doesn't move.

To set a trailing stop loss order, see the Buy and Hold with a Trailing Stop Bootcamp lesson.

Multi-Leg Orders

Multi-leg orders are orders that contain multiple sub-orders. Examples of multi-leg orders include Option strategies like spreads , straddles , and strangles . You can manually implement other types of multi-leg orders with the built-in order types.

Trading and Orders

Position Sizing

Introduction

LEAN provides several methods to help you set specific portfolio weights for assets. These methods account for lot size and pre-calculated order fees.

Single Asset Targets

The `SetHoldings` method calculates the number of asset units to purchase based on the portfolio weight you provide and then submits market orders. This method provides a quick way to set up a portfolio with a set of weights for assets. If you already have holdings, you may want to liquidate the existing holdings first to free up buying power.

```py
 # Allocate 50% of buying power to IBM
self.SetHoldings("IBM", 0.5)

# Allocate 50% of portfolio value to IBM, but liquidate other holdings first
self.SetHoldings("IBM", 0.5, True)

# Provide a tag and order properties to the SetHoldings method
self.SetHoldings(symbol, weight, liquidate_existing_holdings, tag, order_properties)
```

If the percentage you provide translates to an order quantity of 0, the `SetHoldings` method doesn't place an order and doesn't log anything.

Multiple Asset Targets

When you trade a weighted basket of assets, sometimes you must intelligently scale down existing positions before increasing allocations to other assets. If you call the `SetHoldings` method with a list of `PortfolioTarget` objects, LEAN sorts the orders based on your position delta and then places the orders that reduce your position size in an asset before it places orders that increase your position size in an asset. When you call the `SetHoldings` method with a list of `PortfolioTarget` objects, the decimal you pass to the `PortfolioTarget` constructor represents the portfolio weight. In this situation, don't use the `PortfolioTarget.Percent` method.

```py
# Purchase a portfolio of targets, processing orders intelligently.
self.SetHoldings([PortfolioTarget("SPY", 0.8), PortfolioTarget("IBM", 0.2)])
```

Calculate Order Quantities

The `SetHoldings` method uses market order to reach the target portfolio weight you provide. If you want to use a different order type, you need to specify the quantity to trade. To calculate the number of units to trade based on a portfolio weight, call the `CalculateOrderQuantity` method. The method calculates the quantity based on the current price of the asset and adjusts it for the fee model of the security. The target weight you provide is an unleveraged value. For instance, if you have 2x leverage and request a 100% weight, the method calculates the quantity that uses

half of your available margin.

```py
 # Calculate the fee-adjusted quantity of shares with given buying power
quantity = self.CalculateOrderQuantity("IBM", 0.4)
```

Buying Power Buffer

If you place a market on open order near the market close, the market can gap overnight to a worse open price. If the gap is large enough against your trade direction, you may not have sufficient buying power at the open to fill your trade. To ensure a high probability of order fills through market gaps and discontinuities, the `SetHoldings` method assumes a 2.5% cash buffer. If LEAN rejects your orders due to buying power, widen the cash buffer through the algorithm `Settings` property.

```py
# Set the cash buffer to 5%
self.Settings.FreePortfolioValuePercentage = 0.05

# Set the cash buffer to $10,000
self.Settings.FreePortfolioValue = 10000
```

If you use `FreePortfolioValuePercentage` , you must set it in the `Initialize` or `PostInitialize` event handler. If you use `FreePortfolioValue` , you must set it after the `PostInitialize` event handler.

Trading and Orders

Liquidating Positions

Introduction

The `Liquidate` method lets you liquidate individual assets or your entire portfolio. The method creates market orders to close positions and returns the IDs of the liquidation orders. If you have pending open orders for the security when you call `Liquidate` , LEAN tries to cancel them. The `Liquidate` method works for all asset classes, except Crypto. To liquidate Crypto positions, see Crypto Trades .

Liquidate Individual Positions

To liquidate your holdings in an individual security, call the `Liquidate` method and provide a ticker or `Symbol` .

```
PY
# Liquidate all IBM in your portfolio
order_ids = self.Liquidate("IBM")
```

You can pass an order tag to the `Liquidate` method.

```
PY
self.Liquidate("AAPL", "Liquidated")
```

Liquidate All Positions

To liquidate all of the positions in your portfolio, call the `Liquidate` method without any ticker of `Symbol` arguments.

```
PY
// Liquidate your entire portfolio
order_ids = self.Liquidate()
```

You can pass an order tag to the `Liquidate` method.

```
PY
self.Liquidate(tag = "Liquidated")
```

Enable and Disable Liquidations

By default, the `Liquidate` method is functional. To enable and disable it, set the `Settings.LiquidateEnabled` property.

```
PY
# Disable liquidations
self.Settings.LiquidateEnabled = False

# Enable liquidations
self.Settings.LiquidateEnabled = True
```

Market Closed Considerations

If you liquidate your positions when the market is closed, LEAN converts the orders into market on open orders. If your brokerage doesn't support market on open orders , the order is invalid.

Trading and Orders

Crypto Trades

Introduction

All fiat and Crypto currencies are individual assets. When you buy a pair like BTCUSD, you trade USD for BTC. In this case, LEAN removes some USD from your portfolio cash book and adds some BTC. The virtual pair BTCUSD represents your position in the trade, but the virtual pair doesn't actually exist. It simply represents an open trade.

Place Trades

When you place Crypto trades, don't use the `CalculateOrderQuantity` or `SetHoldings` methods. Instead, calculate the order quantity based on the currency amounts in your cash book and place manual orders.

The following code snippet demonstrates how to allocate 90% of your portfolio to BTC.

```py
def OnData(self, data: Slice):
    self.set_crypto_holdings(self.symbol, .9)

def set_crypto_holdings(self, symbol, percentage):
    crypto = self.Securities[symbol]
    base_currency = crypto.BaseCurrency

    # Calculate the target quantity in the base currency
    target_quantity = percentage * (self.Portfolio.TotalPortfolioValue - self.Settings.FreePortfolioValue)
/ base_currency.ConversionRate
    quantity = target_quantity - base_currency.Amount

    # Round down to observe the lot size
    lot_size = crypto.SymbolProperties.LotSize
    quantity = round(quantity / lot_size) * lot_size

    if self.is_valid_order_size(crypto, quantity):
        self.MarketOrder(symbol, quantity)

# Brokerages have different order size rules
# Binance considers the minimum volume (price x quantity):
def is_valid_order_size(self, crypto, quantity):
    return abs(crypto.Price * quantity) > crypto.SymbolProperties.MinimumOrderSize
```

The preceding example doesn't take into account order fees. You can add a 0.1% buffer to accommodate it.

The following example demonstrates how to form an equal-weighted Crypto portfolio and stay within the cash buffer .

```py
def OnData(self, data: Slice):
    percentage = (1 - self.Settings.FreePortfolioValuePercentage) / len(self.symbols);
    for symbol in self.symbols:
        self.set_crypto_holdings(symbol, percentage)
```

You can replace the `self. Settings.FreePortfolioValuePercentage` for a class variable (e.g. `self.cash_buffer`).

When you place Crypto trades, ensure you have a sufficient balance of the base or quote currency before each trade. If you hold multiple assets and you want to put all of your capital into BTCUSD, you need to first convert all your non-

BTC assets into USD and then purchase BTCUSD.

For a full example of placing crypto trades, see the BasicTemplateCryptoAlgorithm .

Liquidate Positions

If you use the `Liquidate` method to liquidate a Crypto position, it only liquidates the quantity of the virtual pair. Since the virtual pair BTCUSD may not represent all of your BTC holdings, don't use the `Liquidate` method to liquidate Crypto positions. Instead, calculate the order quantity based on the currency amounts in your cash book and place manual orders. The following code snippet demonstrates how to liquidate a BTCUSD position.

```
PY
def OnData(self, data: Slice):
    self.liquidate_crypto(self.symbol)

def liquidate_crypto(self, symbol):
    crypto = self.Securities[symbol]
    base_currency = crypto.BaseCurrency

    # Avoid negative amount after liquidate
    quantity = min(crypto.Holdings.Quantity, base_currency.Amount)

    # Round down to observe the lot size
    lot_size = crypto.SymbolProperties.LotSize;
    quantity = (round(quantity / lot_size) - 1) * lot_size

    if self.IsValidOrderSize(crypto, quantity):
        self.MarketOrder(symbol, -quantity)
```

The order fees don't respect the lot size. When you try to liquidate a position, the absolute value of the base currency quantity can be less than the lot size and greater than zero. In this case, your algorithm holds a position that you can't liquidate and `self. Portfolio[symbol].Invested` is `True` . The following code snippet demonstrates how to determine if you can liquidate a position:

```
PY
def OnData(self, data: Slice):
    crypto = self.Securities[self.symbol]
    if abs(crypto.Holdings.Quantity) > crypto.SymbolProperties.LotSize:
        self.liquidate_crypto(self.symbol)
```

Trading and Orders

Option Strategies

You can implement any of the following Option strategies in an algorithm. Click one to learn more.

Bear Call Spread

Bear Put Spread

Bull Call Spread

Bull Put Spread

Call Butterfly

Put Butterfly

Call Calendar Spread

Put Calendar Spread

Covered Call

Covered Put

Iron Butterfly

Iron Condor

Protective Call

Protective Put

Protective Collar

Straddle

Strangle

See Also

Equity Options
Future Options
Index Options

Option Strategies

Bear Call Spread

Introduction

Bear call spread , also known as **short call spread** , consists of selling an ITM call and buying an OTM call. Both calls have the same underlying Equity and the same expiration date. The ITM call serves as a hedge for the OTM call. The bear call spread profits from a drop in underlying asset price.

Implementation

Follow these steps to implement the bear call spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 5)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.IncludeWeeklys().Strikes(-15, 15).Expiration(timedelta(0), timedelta(31))
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the furthest expiry date of the contracts
    expiry = sorted(chain, key = lambda x: x.Expiry, reverse=True)[0].Expiry

    # Select the call Option contracts with the furthest expiry
    calls = [i for i in chain if i.Expiry == expiry and i.Right == OptionRight.Call]
    if len(calls) == 0: return

    # Select the ITM and OTM contract strike prices from the remaining contracts
    call_strikes = sorted([x.Strike for x in calls])
    itm_strike = call_strikes[0]
    otm_strike = call_strikes[-1]
```

3. In the `OnData` method, call the `OptionStrategies.BearCallSpread` method and then submit the order.

```
PY
option_strategy = OptionStrategies.BearCallSpread(self.symbol, itm_strike, otm_strike, expiry)
self.Buy(option_strategy, 1)
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

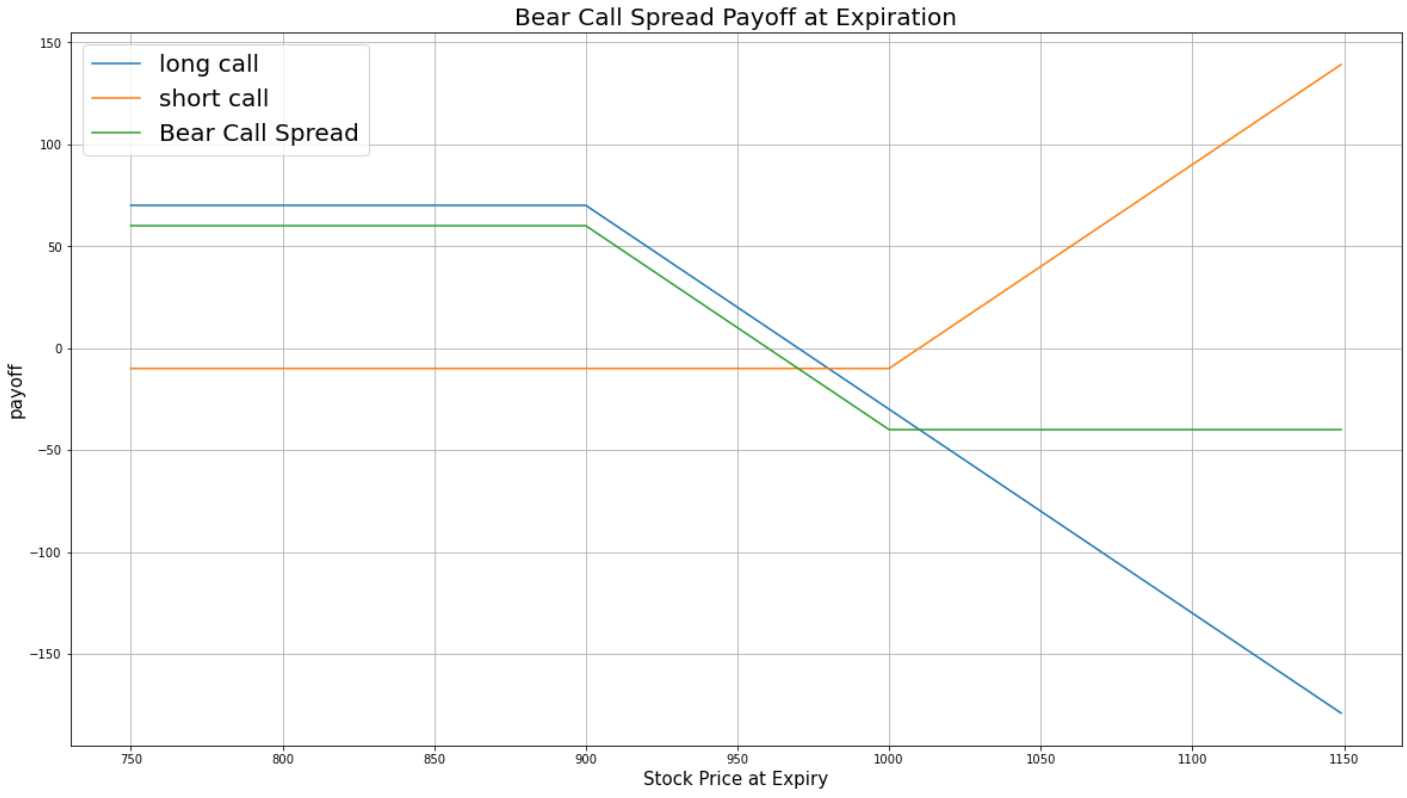
The bear call spread is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
P_T &= (C_T^{OTM} - C_T^{ITM} + C_0^{ITM} - C_0^{OTM}) \times m - fee
\end{aligned}
$$

where

C_T^{OTM}	=	OTM call value at time T
C_T^{ITM}	=	ITM call value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM call strike price
K^{ITM}	=	ITM call strike price
P_T	=	Payout total at time T
C_0^{ITM}	=	ITM call value at position opening (debit paid)
C_0^{OTM}	=	OTM call value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is the net credit you receive from opening the trade, $C_0^{ITM} - C_0^{OTM}$. If the price declines, both calls

expire worthless.

The maximum loss is $K^{OTM} - K^{ITM} + C_0^{ITM} - C_0^{OTM}$, which occurs when the underlying price is above the strike prices of both call Option contracts.

If the Option is American Option, there is a risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

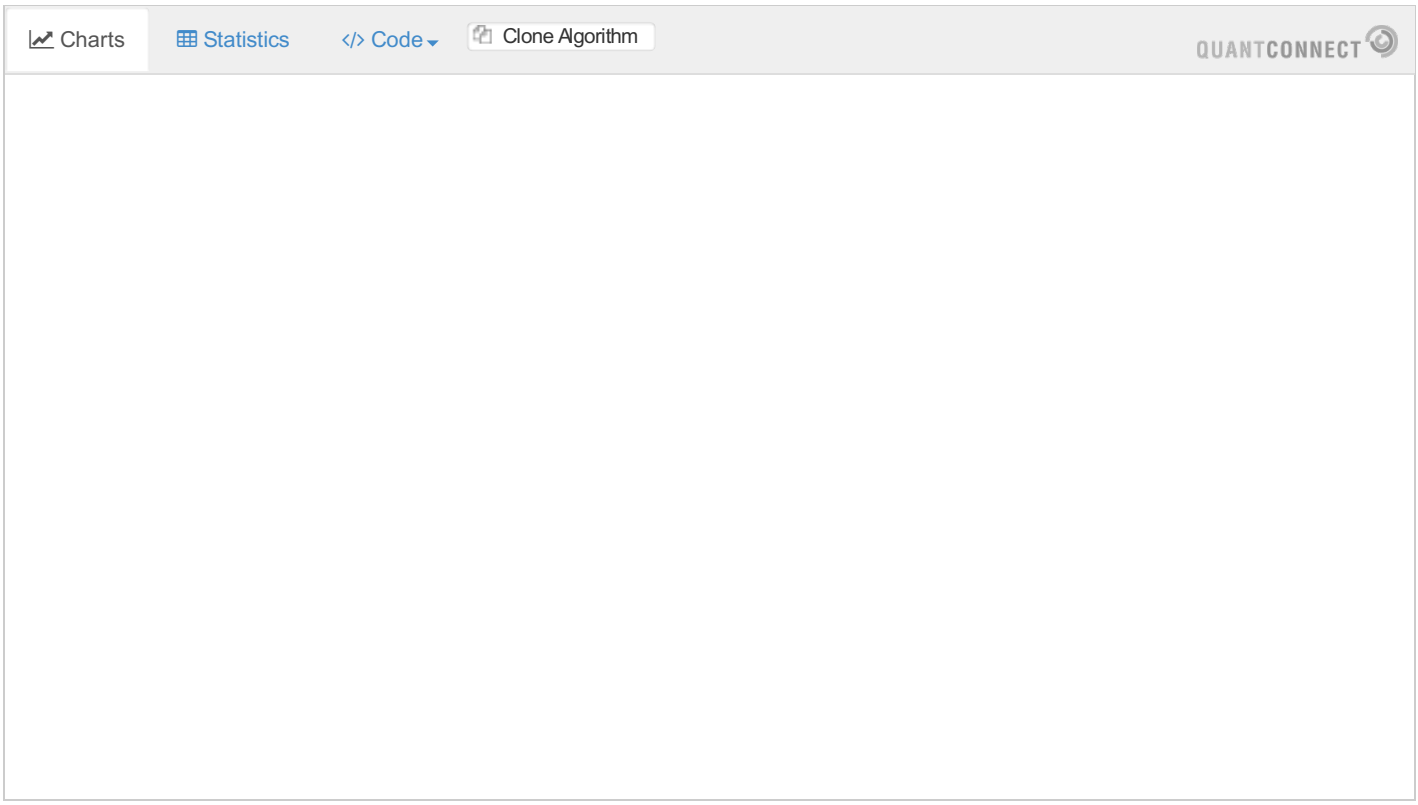
Asset	Price ($)	Strike ($)
OTM call	4.40	835.00
ITM call	36.80	767.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$\begin{aligned} C_T^{OTM} &= (S_T - K^{OTM})^+ \\ &= (829.08 - 835.00)^+ \\ &= 0 \\ C_T^{ITM} &= (S_T - K^{ITM})^+ \\ &= (829.08 - 767.50)^+ \\ &= 61.58 \\ P_T &= (C_T^{OTM} - C_T^{ITM} + C_0^{ITM} - C_0^{OTM}) \times m - fee \\ &= (0 - 61.58 + 36.80 - 4.40) \times 100 - 1.00 \times 2 \\ &= -2920 \end{aligned}$$

So, the strategy losses $2,920.

The following algorithm implements a bear call spread Option strategy:



Demonstration Algorithm
IndexOptionBearCallSpreadAlgorithm.py Python

Option Strategies

Bear Put Spread

Introduction

Bear put spread , also known as **short put spread** , consists of buying an ITM put and selling an OTM put. Both puts have the same underlying Equity and the same expiration date. The OTM put serves as a hedge for the ITM put. The bear put spread profits from a decline in underlying asset price.

Implementation

Follow these steps to implement the bear put spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 5)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.IncludeWeeklys().Strikes(-15, 15).Expiration(timedelta(0), timedelta(31))
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the furthest expiry date of the contracts
    expiry = sorted(chain, key = lambda x: x.Expiry, reverse=True)[0].Expiry

    # Select the put Option contracts with the furthest expiry
    puts = [i for i in chain if i.Expiry == expiry and i.Right == OptionRight.Put]
    if len(puts) == 0: return

    # Select the ITM and OTM contract strike prices from the remaining contracts
    put_strikes = sorted([x.Strike for x in puts])
    otm_strike = put_strikes[0]
    itm_strike = put_strikes[-1]
```

3. In the `OnData` method, call the `OptionStrategies.BearPutSpread` method and then submit the order.

```
option_strategy = OptionStrategies.BearPutSpread(self.symbol, itm_strike, otm_strike, expiry)
self.Buy(option_strategy, 1)
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

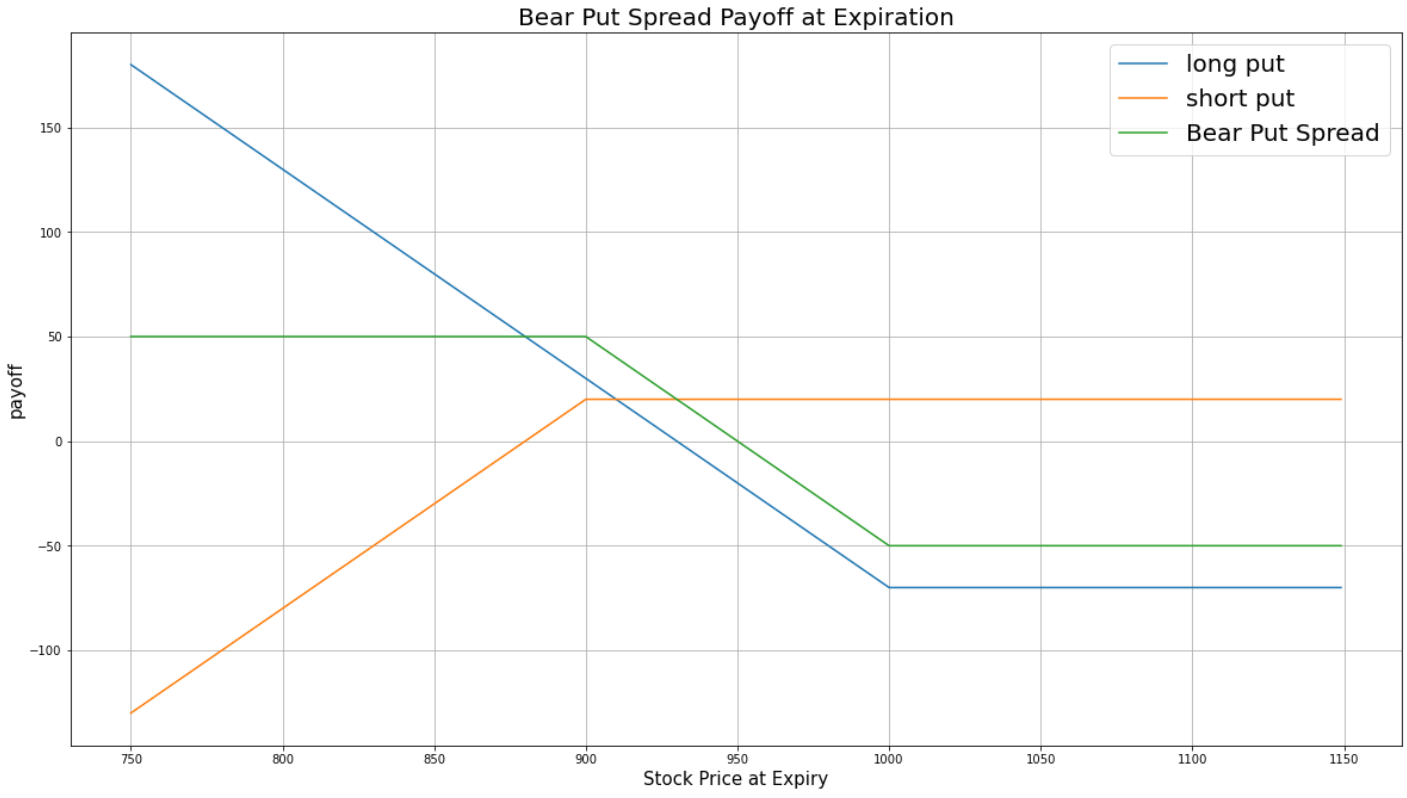
The bear put spread is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
P_T &= (P_T^{ITM} - P_T^{OTM} + P_0^{OTM} - P_0^{ITM}) \times m - fee
\end{aligned}
$$

where

P_T^{OTM}	=	OTM put value at time T
P_T^{ITM}	=	ITM put value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM put strike price
K^{ITM}	=	ITM put strike price
P_T	=	Payout total at time T
P_0^{ITM}	=	ITM put value at position opening (debit paid)
P_0^{OTM}	=	OTM put value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is $K^{ITM} - K^{OTM} + P_0^{OTM} - P_0^{ITM}$. If the underlying price is below than the strike prices of both put

Option contracts, they are worth $(K - S_T)$ at expiration.

The maximum loss is the net debit you paid to open the position, $P_0^{OTM} - P_0^{ITM}$.

If the Option is American Option, there is a risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

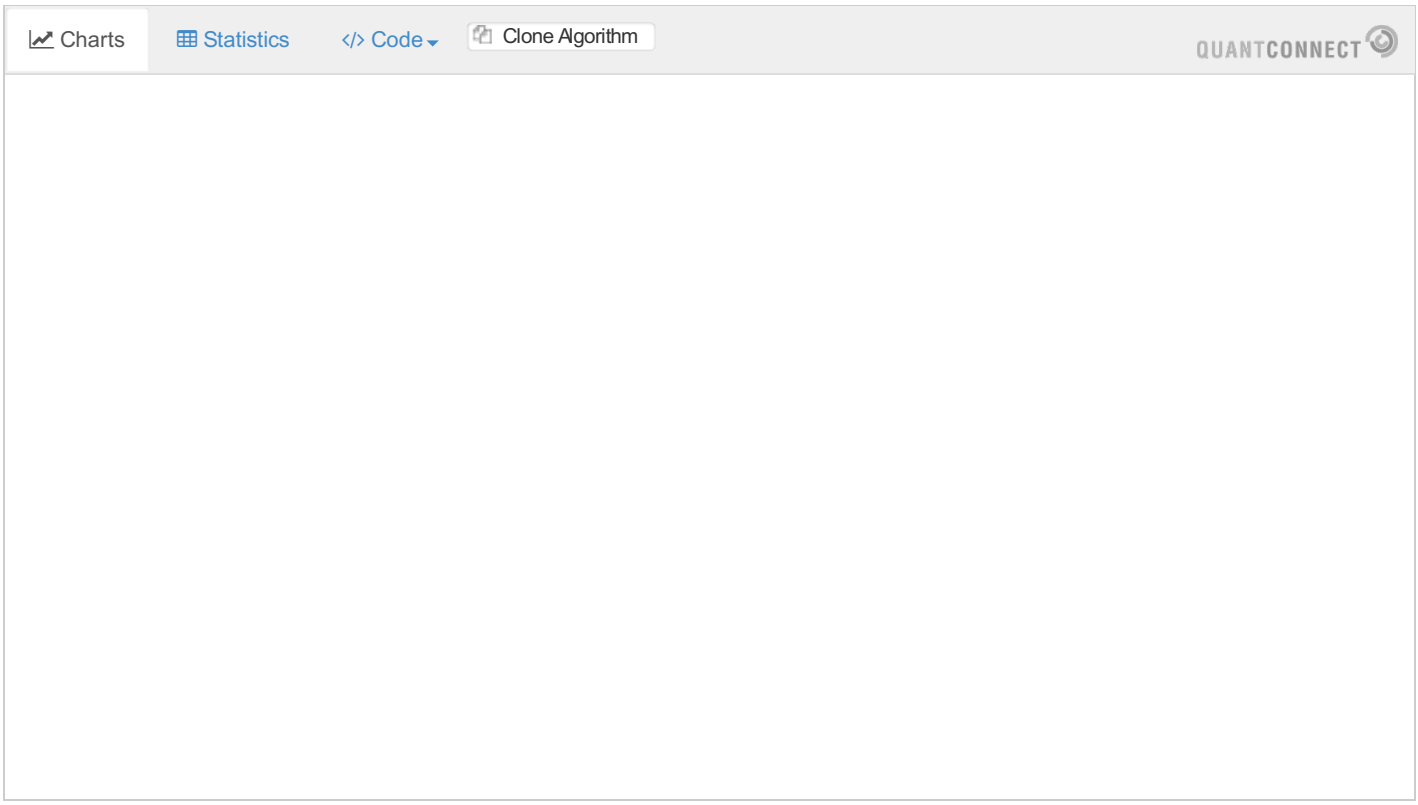
Asset	**Price ($)**	**Strike ($)**
OTM put	4.60	767.50
ITM put	40.00	835.00
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
&= (767.50 - 829.08)^+ \\
&= 0 \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
&= (835.00 - 829.08)^+ \\
&= 5.92 \\
P_T &= (P_T^{ITM} - P_T^{OTM} + P_0^{OTM} - P_0^{ITM}) \times m - fee \\
&= (5.92 - 0 + 4.60 - 40.00) \times 100 - 1.00 \times 2 \\
&= -2950
\end{aligned}
$$

So, the strategy losses $2,950.

The following algorithm implements a bear put spread strategy:



Demonstration Algorithm
IndexOptionBearPutSpreadAlgorithm.py Python

Option Strategies

Bull Call Spread

Introduction

Bull call spread , also known as **long call spread** , consists of buying an ITM call and selling an OTM call. Both calls have the same underlying Equity and the same expiration date. The OTM call serves as a hedge for the ITM call. The bull call spread profits from a rise in underlying asset price.

Implementation

Follow these steps to implement the bull call spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 5)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.IncludeWeeklys().Strikes(-15, 15).Expiration(timedelta(0), timedelta(31))
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the furthest expiration date of the contracts
    expiry = sorted(chain, key = lambda x: x.Expiry, reverse=True)[0].Expiry

    # Select the call Option contracts with the furthest expiry
    calls = [i for i in chain if i.Expiry == expiry and i.Right == OptionRight.Call]
    if len(calls) == 0: return

    # Select the ITM and OTM contract strike prices from the remaining contracts
    call_strikes = sorted([x.Strike for x in calls])
    itm_strike = call_strikes[0]
    otm_strike = call_strikes[-1]
```

3. In the `OnData` method, call the `OptionStrategies.BullCallSpread` method and then submit the order.

```
PY
option_strategy = OptionStrategies.BullCallSpread(self.symbol, itm_strike, otm_strike, expiry)
self.Buy(option_strategy, 1)
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

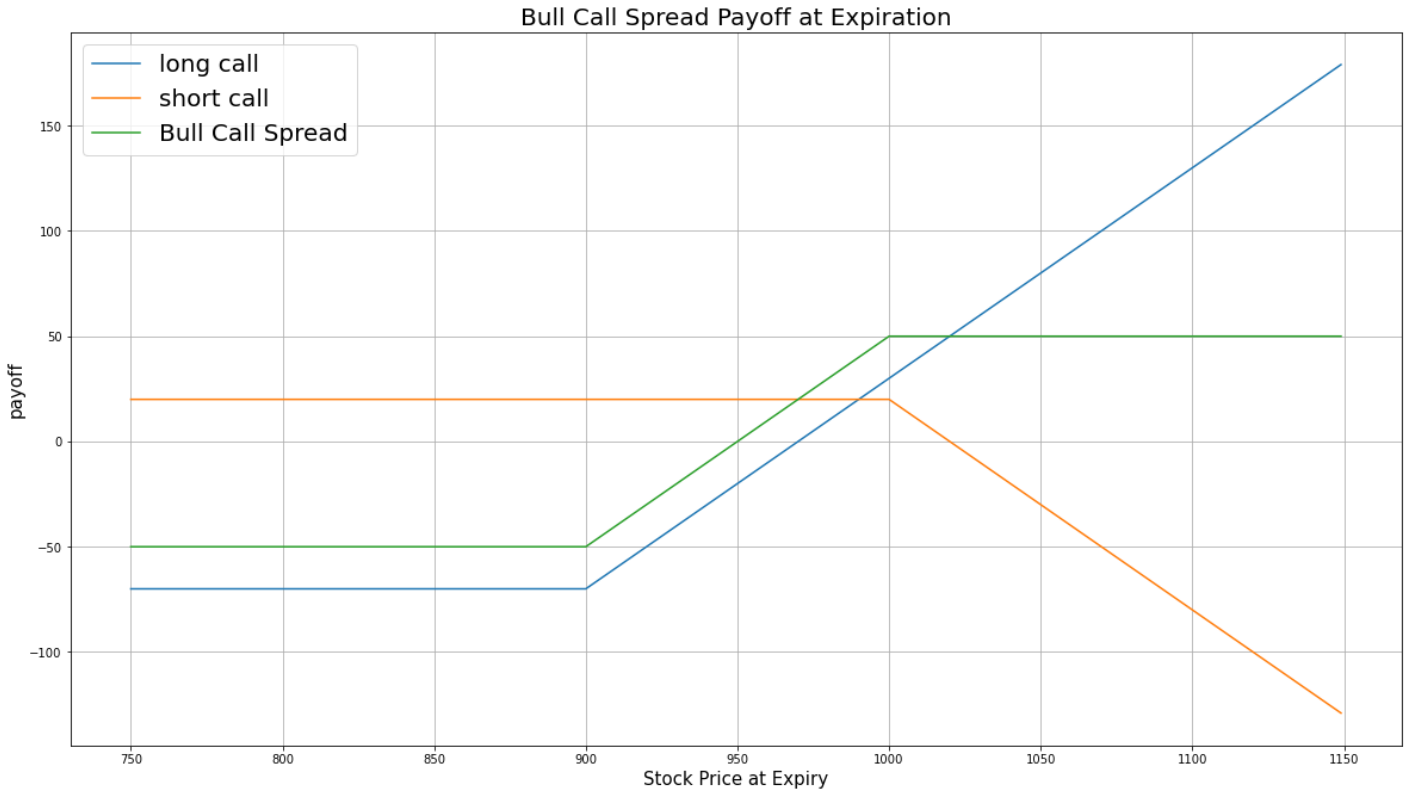
The bull call spread is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
P_T &= (C_T^{ITM} - C_T^{OTM} + C_0^{OTM} - C_0^{ITM}) \times m - fee
\end{aligned}
$$

where

C_T^{OTM}	=	OTM call value at time T
C_T^{ITM}	=	ITM call value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM call strike price
K^{ITM}	=	ITM call strike price
P_T	=	Payout total at time T
C_0^{ITM}	=	ITM call value at position opening (debit paid)
C_0^{OTM}	=	OTM call value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is $K^{OTM} - K^{ITM} + C_0^{OTM} - C_0^{ITM}$. If the underlying price increases to exceed both strikes at expiration,

both calls are worth ($S_T - K$) at expiration.

The maximum loss is the net debit you paid to open the position, $C_0^{OTM} - C_0^{ITM}$.

If the Option is American Option, there is a risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

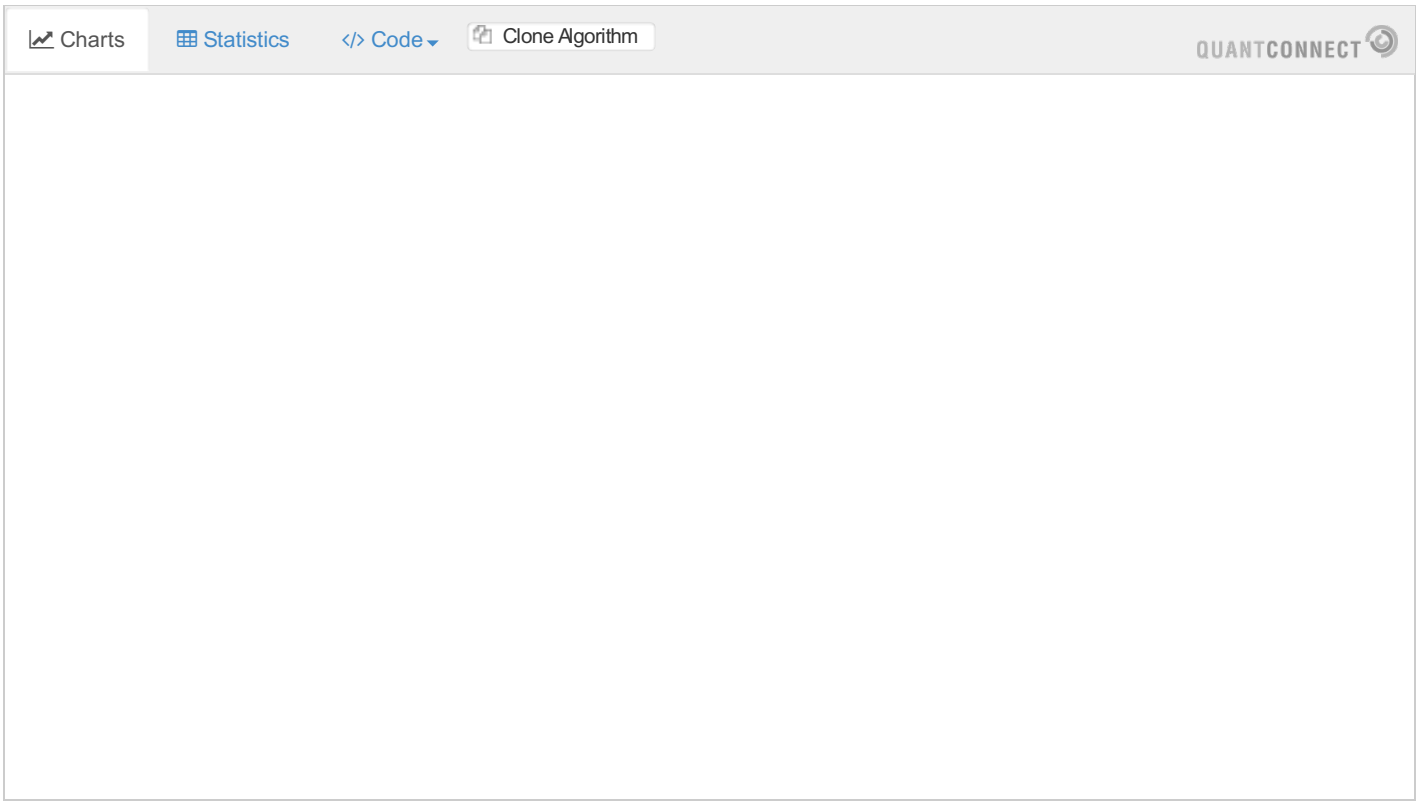
Asset	**Price ($)**	**Strike ($)**
OTM call	3.00	835.00
ITM call	41.00	767.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
&= (829.08 - 835.00)^+ \\
&= 0 \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
&= (829.08 - 767.50)^+ \\
&= 61.58 \\
P_T &= (C_T^{ITM} - C_T^{OTM} + C_0^{OTM} - C_0^{ITM}) \times m - fee \\
&= (61.58 - 0 + 3.00 - 41.00) \times 100 - 1.00 \times 2 \\
&= 2356
\end{aligned}
$$

So, the strategy profits $2,356.

The following algorithm implements a bull call spread strategy:



Demonstration Algorithm
IndexOptionBullCallSpreadAlgorithm.py Python

Option Strategies

Bull Put Spread

Introduction

Bull put spread , also known as **long put spread** , consists of buying an OTM put and selling an ITM put. Both puts have the same underlying Equity and the same expiration date. The OTM put serves as a hedge for the ITM put. The bull put spread profits from a rise in underlying asset price.

Implementation

Follow these steps to implement the bull put spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 5)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.IncludeWeeklys().Strikes(-15, 15).Expiration(timedelta(0), timedelta(31))
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the furthest expiration date of the contracts
    expiry = sorted(chain, key = lambda x: x.Expiry, reverse=True)[0].Expiry

    # Select the put Option contracts with the furthest expiry
    puts = [i for i in chain if i.Expiry == expiry and i.Right == OptionRight.Put]
    if len(puts) == 0: return

    # Select the ITM and OTM contract strikes from the remaining contracts
    put_strikes = sorted([x.Strike for x in puts])
    otm_strike = put_strikes[0]
    itm_strike = put_strikes[-1]
```

3. In the `OnData` method, call the `OptionStrategies.BullPutSpread` method and then submit the order.

```
PY
option_strategy = OptionStrategies.BullPutSpread(self.symbol, itm_strike, otm_strike, expiry)
self.Buy(option_strategy, 1)
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

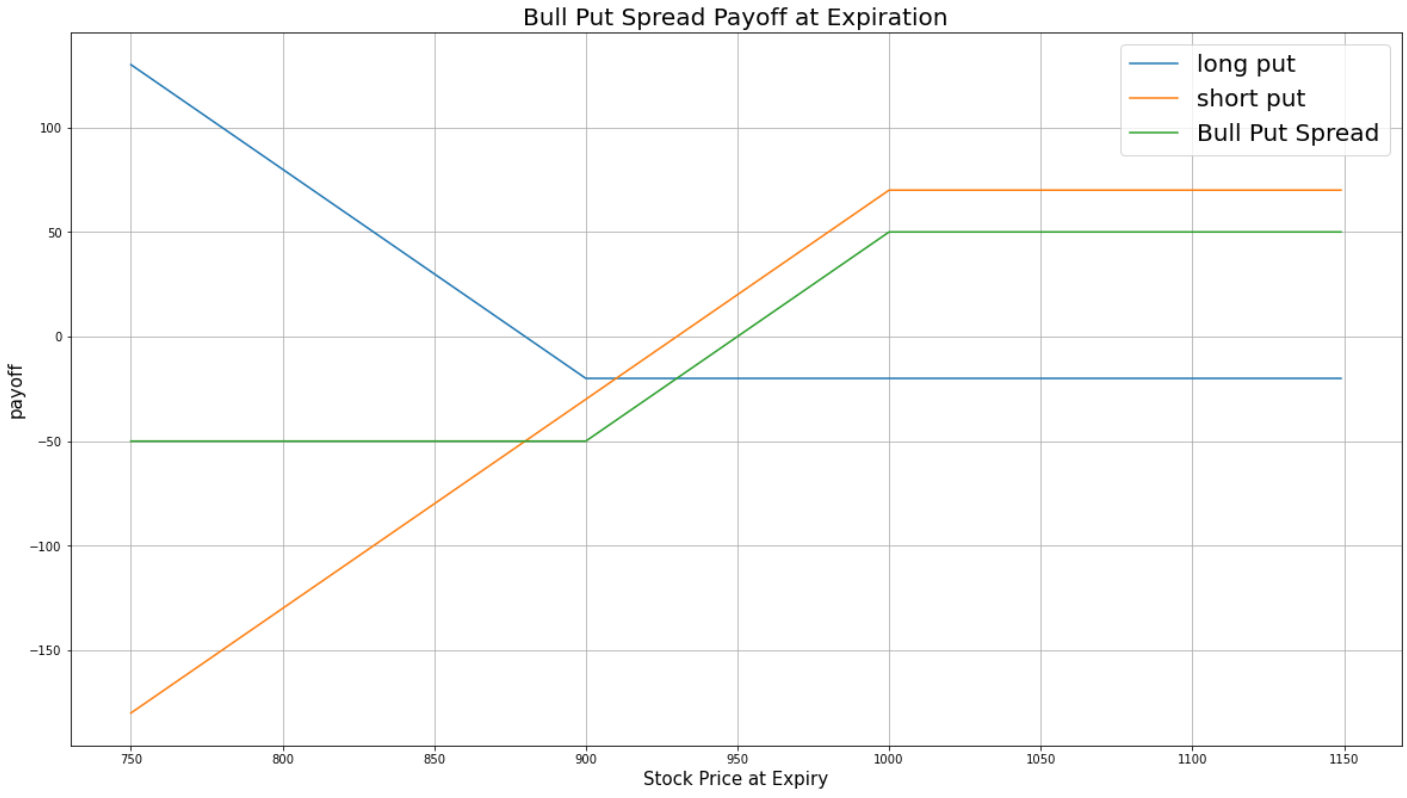
This is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
P_T &= (P_T^{OTM} - P_T^{ITM} + P_0^{ITM} - P_0^{OTM}) \times m - fee
\end{aligned}
$$

where

P_T^{OTM}	=	OTM put value at time T
P_T^{ITM}	=	ITM put value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM put strike price
K^{ITM}	=	ITM put strike price
P_T	=	Payout total at time T
P_0^{ITM}	=	ITM put value at position opening (credit received)
P_0^{OTM}	=	OTM put value at position opening (debit paid)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is the net credit you received when opening the position, $P_0^{ITM} - P_0^{OTM}$. If the underlying price is

higher than the strike prices of both put contracts at expiration, both puts expire worthless.

The maximum loss is $K^{ITM} - K^{OTM} + P_0^{ITM} - P_0^{OTM}$.

If the Option is American Option, there is a risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

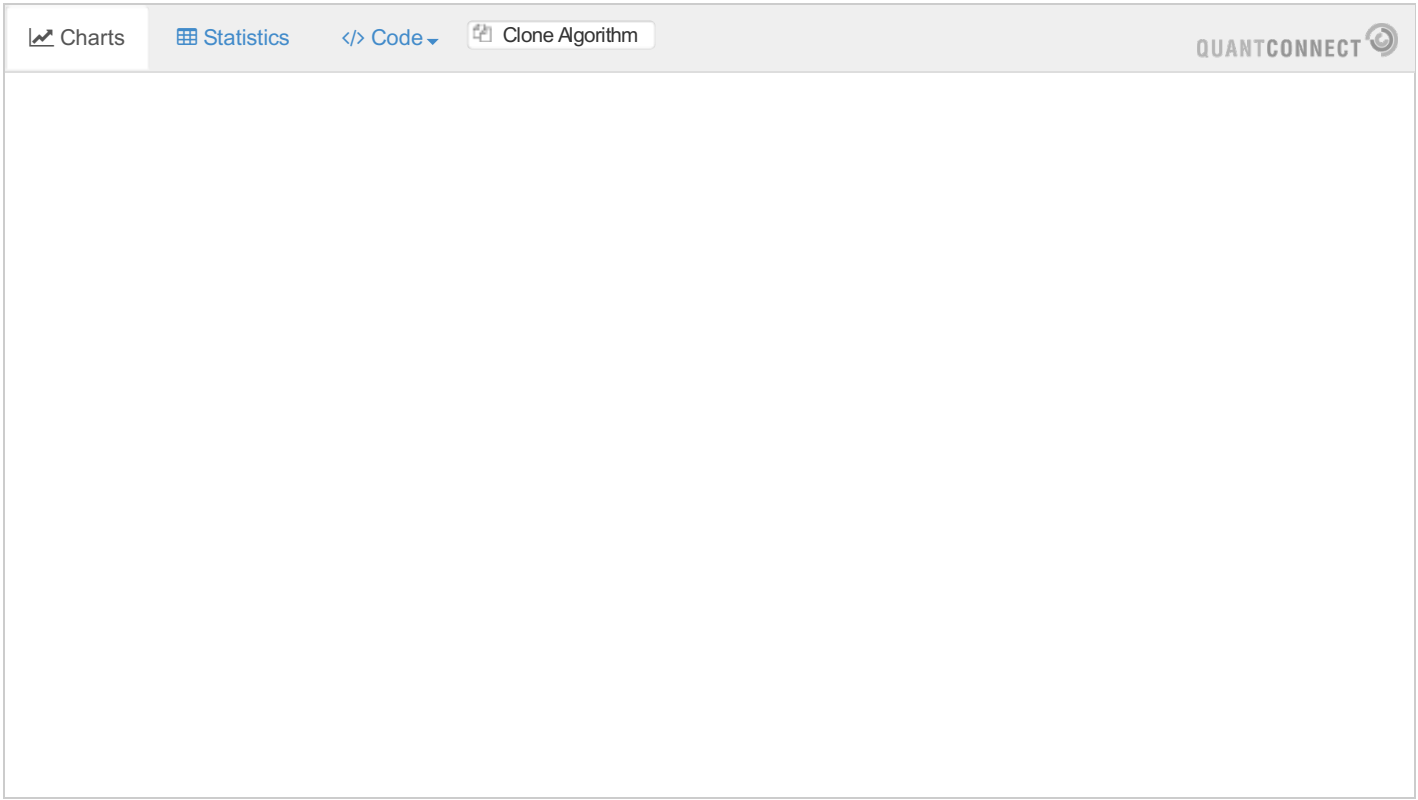
Asset	Price ($)	Strike ($)
OTM put	5.70	767.50
ITM put	35.50	835.00
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{OTM}_T & = & (K^{OTM} - S_T)^{+}\\ & = & (767.50-829.08)^{+}\\ & = & 0\\ P^{ITM}_T & = & (K^{ITM} - S_T)^{+}\\ & = & (835.00-829.08)^{+}\\ & = & 5.92\\ P_T & = & (P^{OTM}_T - P^{ITM}_T + P^{ITM}_0 - P^{OTM}_0)\times m - fee\\ & = & (0-5.92+35.50-5.70)\times100-1.00\times2\\ & = & 2386\\ \end{array}$$

So, the strategy profits $2,386.

The following algorithm implements a bull put spread strategy:



Demonstration Algorithm
IndexOptionBullPutSpreadAlgorithm.py Python

Option Strategies

Call Butterfly

Introduction

The **call butterfly** strategy is the combination of a bull call spread and a bear call spread . In the call butterfly, all of the calls should have the same underlying Equity, the same expiration date, and the same strike price distance between the ITM-ATM and OTM-ATM call pairs. The call butterfly can be long or short.

Long Call Butterfly

The long call butterfly consists of a long ITM call, a long OTM call, and 2 short ATM calls. This strategy profits from low volatility in the underlying Equity price.

Short Call Butterfly

The short call butterfly consists of a short ITM call, a short OTM call, and 2 long ATM calls. This strategy profits from high volatility in the underlying Equity price.

Implementation

Follow these steps to implement the call butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 5)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.IncludeWeeklys().Strikes(-15, 15).Expiration(timedelta(0), timedelta(31))
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the furthest expiry date of the contracts
    expiry = sorted(chain, key = lambda x: x.Expiry, reverse=True)[0].Expiry

    # Select the call Option contracts with the furthest expiry
    calls = [i for i in chain if i.Expiry == expiry and i.Right == OptionRight.Call]
    if len(calls) == 0: return

    # Get the strike prices of the all the call Option contracts
    call_strikes = sorted([x.Strike for x in calls])

    # Get the ATM strike price
    atm_strike = sorted(calls, key=lambda x: abs(x.Strike - chain.Underlying.Price))[0].Strike

    # Get the strike prices for the contracts not ATM
    spread = min(abs(call_strikes[0] - atm_strike), abs(call_strikes[-1] - atm_strike))
    itm_strike = atm_strike - spread
    otm_strike = atm_strike + spread
```

3. In the `OnData` method, call the `OptionStrategies.CallButterfly` method and then submit the order.

```
option_strategy = OptionStrategies.CallButterfly(self.symbol, otm_strike, atm_strike, itm_strike,
expiry)
self.Buy(option_strategy, 1)    # if long call butterfly
self.Sell(option_strategy, 1)   # if short call butterfly
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` and `Sell` methods.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
self.Sell(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

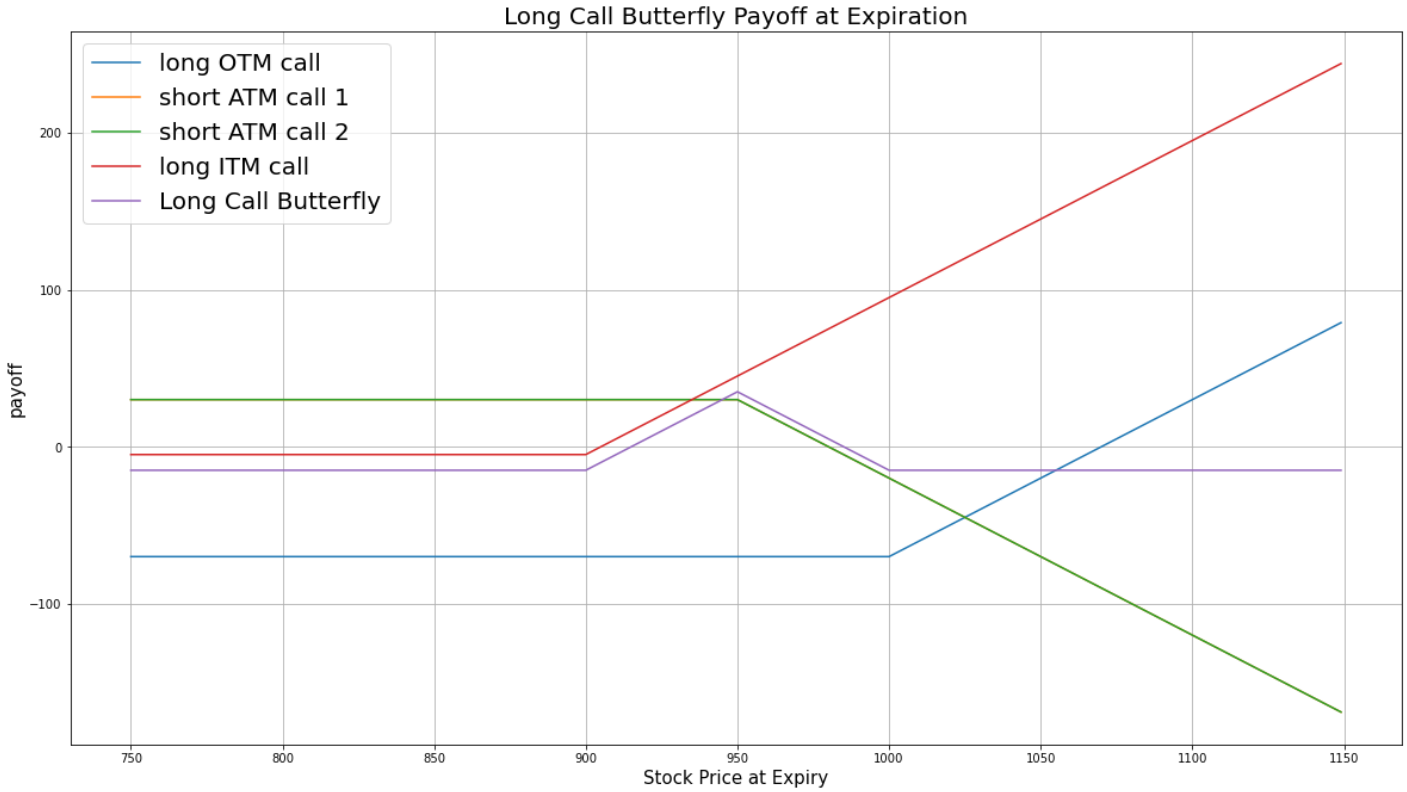
The call butterfly can be long or short.

Long Call Butterfly

The long call butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{OTM})^{+}\\ C^{ITM}_T & = & (S_T - K^{ITM})^{+}\\ C^{ATM}_T & = & (S_T - K^{ATM})^{+}\\ P_T & = & (C^{OTM}_T + C^{ITM}_T - 2\times C^{ATM}_T + 2\times C^{ATM}_0 - C^{ITM}_0 - C^{OTM}_0)\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & C^{ITM}_T & = & \textrm{ITM call value at time T}\\ & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{OTM} & = & \textrm{OTM call strike price}\\ & K^{ITM} & = & \textrm{ITM call strike price}\\ & K^{ATM} & = & \textrm{ATM call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{ITM}_0 & = & \textrm{ITM call value at position opening (debit paid)}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (debit paid)}\\ & C^{ATM}_0 & = & \textrm{OTM call value at position opening (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{ATM} - K^{ITM} + 2\times C^{ATM}_0 - C^{ITM}_0 - C^{OTM}_0$. It occurs when the underlying price is the same price at expiration as it was when opening the position and the payouts of the bull and bear call spreads are at their maximum.

The maximum loss is the net debit paid: $2\times C^{ATM}_0 - C^{ITM}_0 - C^{OTM}_0$. It occurs when the underlying price is less than ITM strike or greater than OTM strike at expiration.

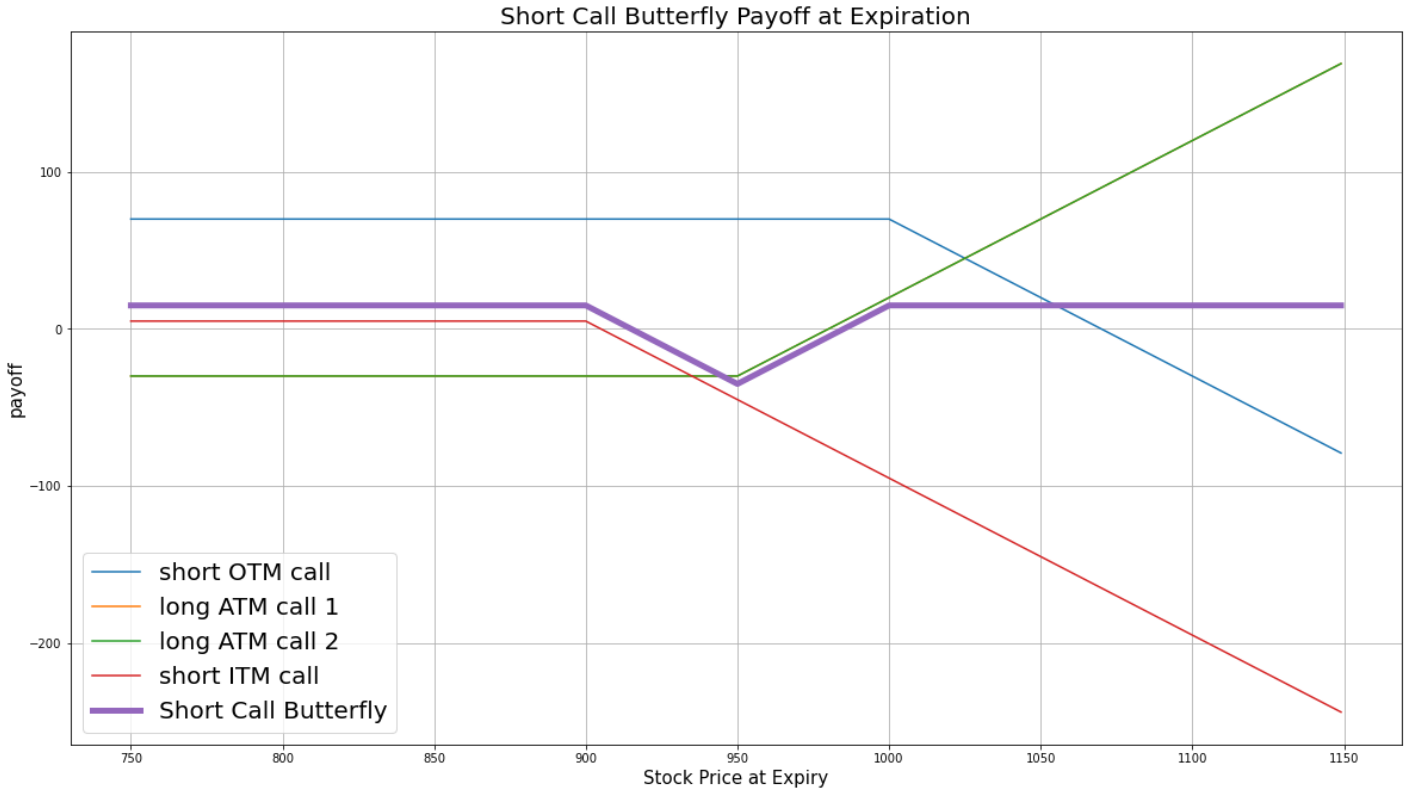
If the Option is an American Option, there is a risk of early assignment on the sold contracts.

Short Call Butterfly

The short call butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{OTM})^{+}\\ C^{ITM}_T & = & (S_T - K^{ITM})^{+}\\ C^{ATM}_T & = & (S_T - K^{ATM})^{+}\\ P_T & = & (2\times C^{ATM}_T - C^{OTM}_T - C^{ITM}_T - 2\times C^{ATM}_0 + C^{ITM}_0 + C^{OTM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & C^{ITM}_T & = & \textrm{ITM call value at time T}\\ & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{OTM} & = & \textrm{OTM call strike price}\\ & K^{ITM} & = & \textrm{ITM call strike price}\\ & K^{ATM} & = & \textrm{ATM call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{ITM}_0 & = & \textrm{ITM call value at position opening (credit received)}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (credit received)}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received: $C^{ITM}_0 + C^{OTM}_0 - 2\times C^{ATM}_0$. It occurs when the underlying price is less than ITM strike or greater than OTM strike at expiration.

The maximum loss is $K^{ATM} - K^{ITM} + C^{ITM}_0 + C^{OTM}_0 - 2\times C^{ATM}_0$. It occurs when the underlying price is at the same level as when you opened the trade.

If the Option is an American Option, there is a risk of early assignment on the sold contracts.

Example

The following table shows the price details of the assets in the long version of the algorithm:

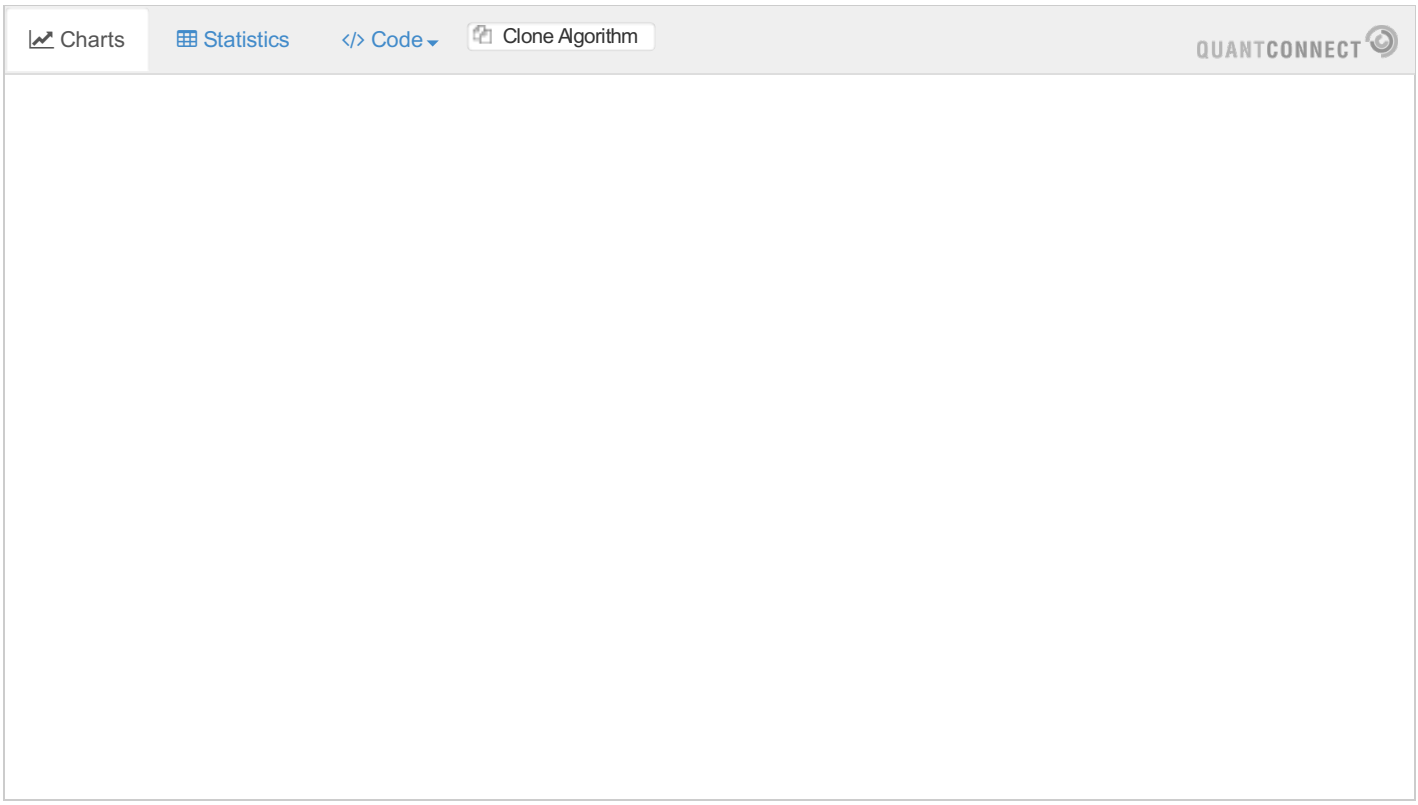
Asset	**Price ($)**	**Strike ($)**
OTM call	4.90	767.50
ATM call	15.00	800.00
ITM call	41.00	832.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{OTM})^{+}\\ & = & (767.50-829.08)^{+}\\ & = & 0\\ C^{ITM}_T & = & (S_T - K^{ITM})^{+}\\ & = & (832.50-829.08)^{+}\\ & = & 3.42\\ C^{ATM}_T & = & (S_T - K^{ATM})^{+}\\ & = & (800.00-829.08)^{+}\\ & = & 0\\ P_T & = & (C^{OTM}_T + C^{ITM}_T - 2\times C^{ATM}_T + 2\times C^{ATM}_0 - C^{ITM}_0 - C^{OTM}_0)\times m - fee\\ & = & (0+3.42-0\times2-4.90-41.00+15.00\times2)\times100-1.00\times4\\ & = & -1252 \end{array}$$

So, the strategy losses $1,252.

The following algorithm implements a long call butterfly Option strategy:



Demonstration Algorithm
IndexOptionCallButterflyAlgorithm.py Python

Option Strategies

Put Butterfly

Introduction

Put butterfly is the combination of a bull put spread and a bear put spread . In this strategy, all the puts have the same underlying stock, the same expiration date, and the strike price distance of ITM-ATM and OTM-ATM put pairs are the same. The put butterfly strategy can be long or short.

Long Put Butterfly

The long put butterfly strategy consists of buying an ITM put, buying an OTM put, and selling 2 ATM puts. This strategy profits from low volatility.

Short Put Butterfly

The short put butterfly strategy consists of selling an ITM put, selling an OTM put, and buying 2 ATM puts. This strategy profits from a drastic change in underlying price.

Implementation

Follow these steps to implement the put butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```py
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 5)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.IncludeWeeklys().Strikes(-15, 15).Expiration(timedelta(0), timedelta(31))
```

2. In the `OnData` method, select strikes and expiration date of the contracts in the strategy legs.

```
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Select an expiry date
    expiry = sorted(chain, key = lambda x: x.Expiry, reverse=True)[0].Expiry

    # Select the put contracts that expire on the selected date
    puts = [i for i in chain if i.Expiry == expiry and i.Right == OptionRight.Put]
    if len(puts) == 0: return

    # Sort the put contracts by their strike prices
    put_strikes = sorted([x.Strike for x in puts])

    # Get the ATM strike price
    atm_strike = sorted(puts, key=lambda x: abs(x.Strike - chain.Underlying.Price))[0].Strike

    # Get the distance between lowest strike price and ATM strike, and highest strike price and ATM
strike.
    # Get the lower value as the spread distance as equidistance is needed for both sides
    spread = min(abs(put_strikes[0] - atm_strike), abs(put_strikes[-1] - atm_strike))

    # Select the strike prices of the strategy legs
    itm_strike = atm_strike + spread
    otm_strike = atm_strike - spread
```

3. In the `OnData` method, call the `OptionStrategies.PutButterfly` method and then submit the order.

```
option_strategy = OptionStrategies.PutButterfly(self.symbol, itm_strike, atm_strike, otm_strike,
expiry)
self.Buy(option_strategy, 1)    # if long put butterfly
self.Sell(option_strategy, 1)   # if short put butterfly
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` and `Sell` methods.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
self.Sell(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

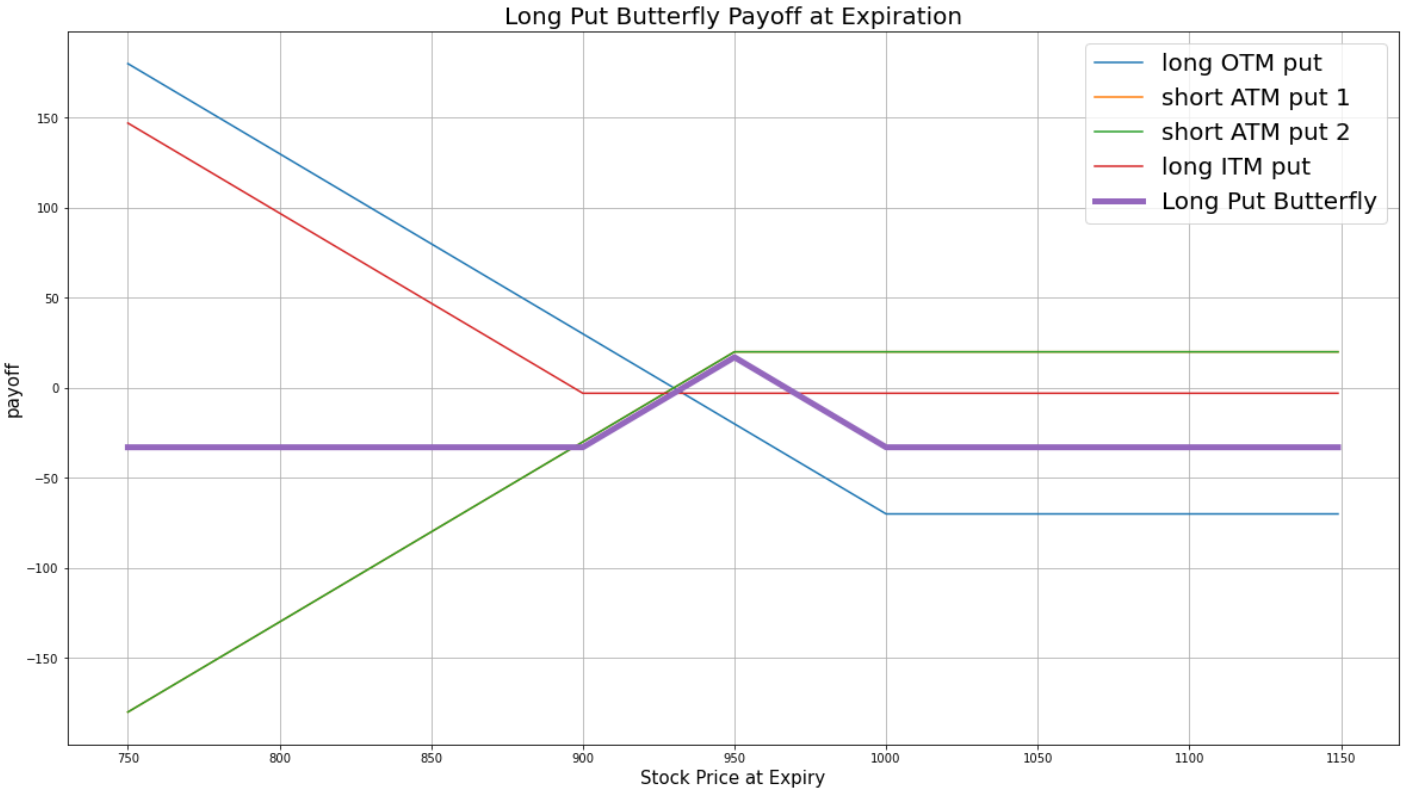
The put butterfly can be long or short.

Long Put Butterfly

The long put butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} P^{OTM}_T &=& (K^{OTM} - S_T)^{+}\\ P^{ITM}_T &=& (K^{ITM} - S_T)^{+}\\ P^{ATM}_T &=& (K^{ATM} - S_T)^{+}\\ P_T &=& (P^{OTM}_T + P^{ITM}_T - 2\times P^{ATM}_T + 2\times P^{ATM}_0 - P^{ITM}_0 - P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & P^{OTM}_T &=& \textrm{OTM put value at time T}\\ & P^{ITM}_T &=& \textrm{ITM put value at time T}\\ & P^{ATM}_T &=& \textrm{ATM put value at time T}\\ & S_T &=& \textrm{Underlying asset price at time T}\\ & K^{OTM} &=& \textrm{OTM put strike price}\\ & K^{ITM} &=& \textrm{ITM put strike price}\\ & K^{ATM} &=& \textrm{ATM put strike price}\\ & P_T &=& \textrm{Payout total at time T}\\ & P^{ITM}_0 &=& \textrm{ITM put value at position opening (debit paid)}\\ & P^{OTM}_0 &=& \textrm{OTM put value at position opening (debit paid)}\\ & P^{ATM}_0 &=& \textrm{ATM put value at position opening (credit received)}\\ & m &=& \textrm{Contract multiplier}\\ & T &=& \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{ATM} - K^{OTM} + 2\times P^{ATM}_0 - P^{ITM}_0 - P^{OTM}_0$. It occurs when the underlying price is the same at expiration as it was when you open the trade. In this case, the payout of the combined bull put and bear put spreads are at their maximum.

The maximum loss is the net debit paid, $2\times P^{ATM}_0 - P^{ITM}_0 - P^{OTM}_0$. It occurs when the underlying price is below the ITM strike price or above the OTM strike price at expiration.

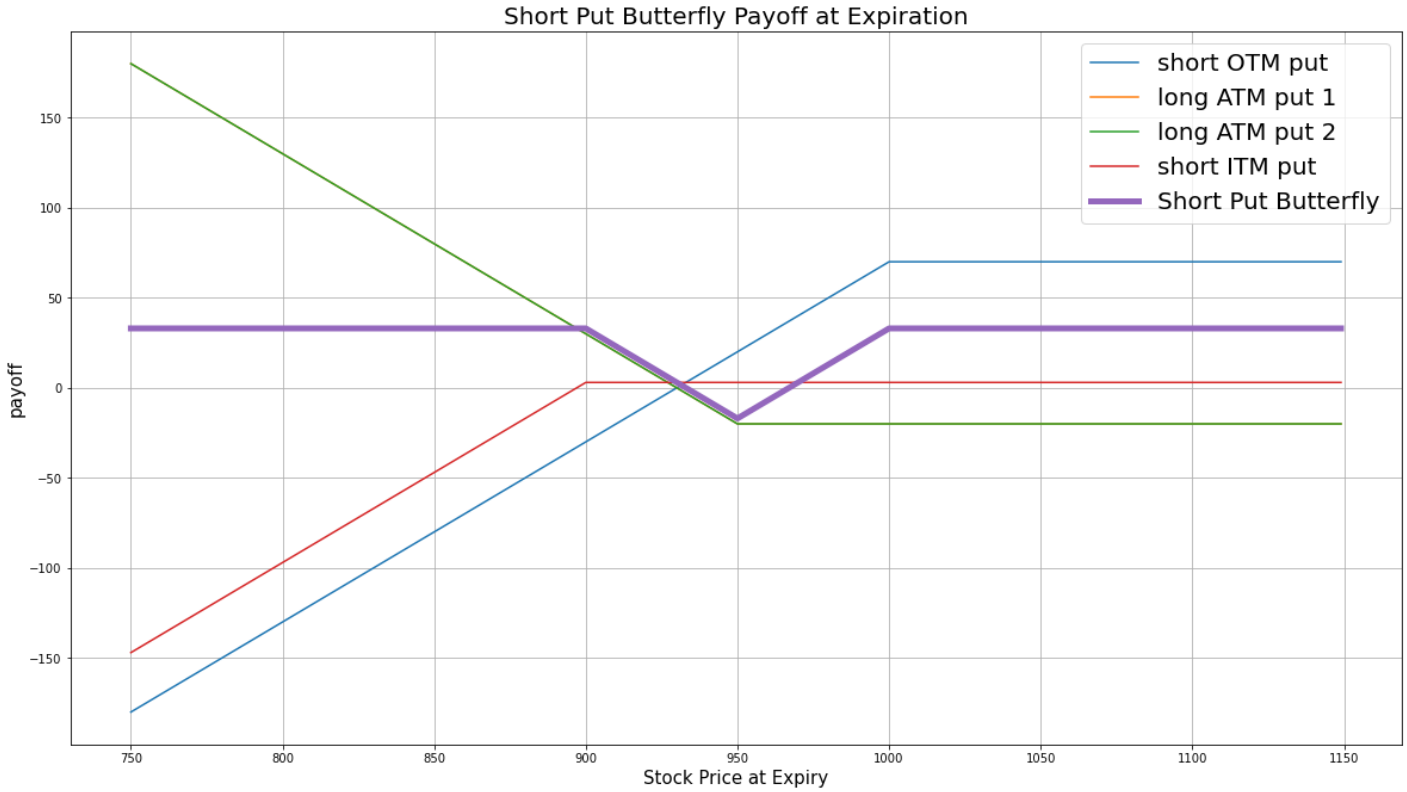
If the Option is American Option, there is risk of early assignment on the sold contracts.

Short Put Butterfly

The short put butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} P^{OTM}_T & = & (K^{OTM} - S_T)^{+}\\ P^{ITM}_T & = & (K^{ITM} - S_T)^{+}\\ P^{ATM}_T & = & (K^{ATM} - S_T)^{+}\\ P_T & = & (2\times P^{ATM}_T - P^{OTM}_T - P^{ITM}_T - 2\times P^{ATM}_0 + P^{ITM}_0 + P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & P^{ITM}_T & = & \textrm{ITM put value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{OTM} & = & \textrm{OTM put strike price}\\ & K^{ITM} & = & \textrm{ITM put strike price}\\ & K^{ATM} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & P^{ITM}_0 & = & \textrm{ITM put value at position opening (credit received)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (credit received)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, $P^{ITM}_0 + P^{OTM}_0 - 2\times P^{ATM}_0$. It occurs when the underlying price is below the ITM strike or above the OTM strike at expiration.

The maximum loss is $K^{ATM} - K^{OTM} - 2\times P^{ATM}_0 + P^{ITM}_0 + P^{OTM}_0$. It occurs when the underlying price at expiration is at the same price as when you opened the trade.

If the Option is American Option, there is risk of early assignment on the sold contracts.

Example

The following table shows the price details of the assets in the long put butterfly algorithm:

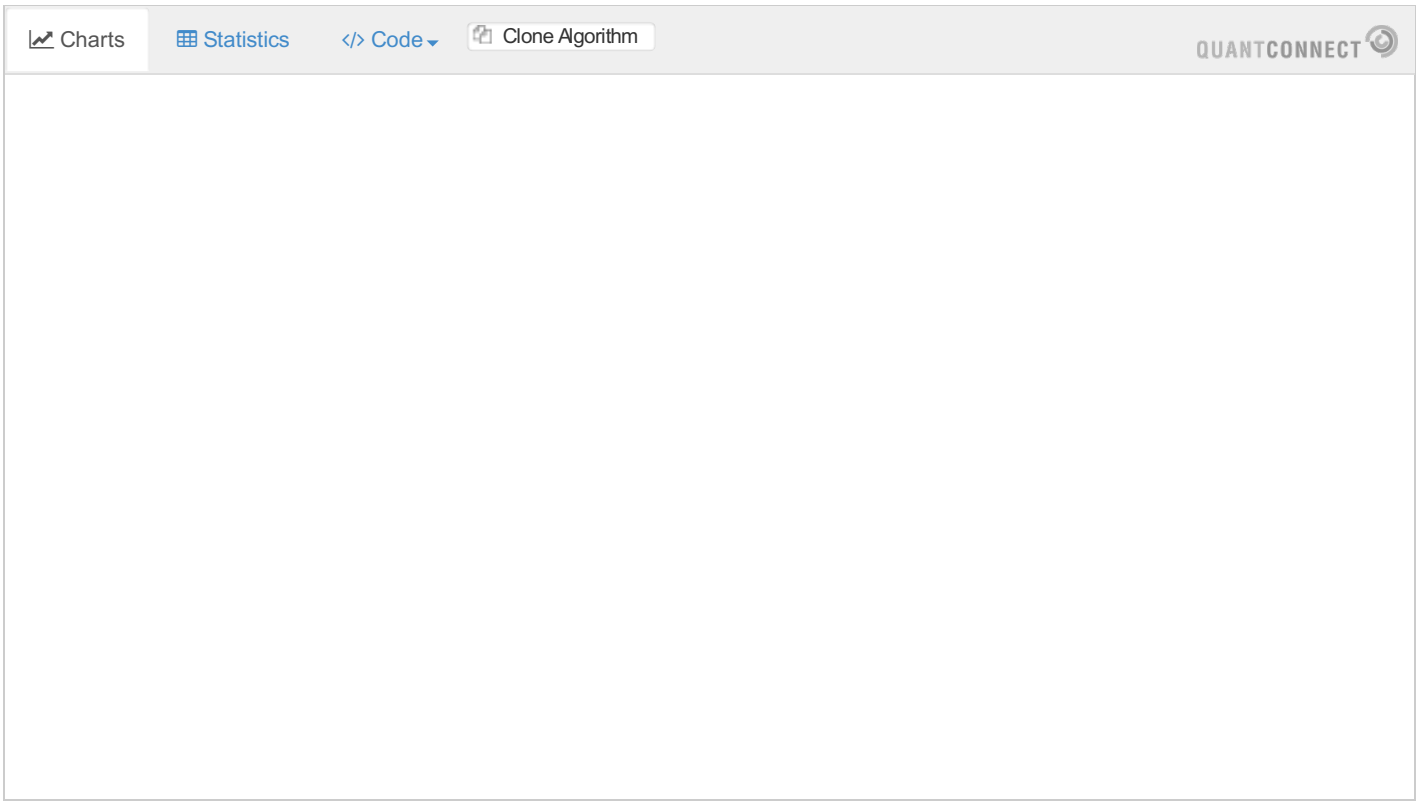
Asset	**Price ($)**	**Strike ($)**
ITM put	37.80	832.50
ATM put	14.70	800.00
OTM put	5.70	767.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{OTM}_T & = & (K^{OTM} - S_T)^{+}\\ & = & (829.08-832.50)^{+}\\ & = & 0\\ P^{ITM}_T & = & (K^{ITM} - S_T)^{+}\\ & = & (829.08-767.50)^{+}\\ & = & 61.58\\ P^{ATM}_T & = & (K^{ATM} - S_T)^{+}\\ & = & (829.08-800.00)^{+}\\ & = & 29.08\\ P_T & = & (P^{OTM}_T + P^{ITM}_T - 2\times P^{ATM}_T + 2\times P^{ATM}_0 - P^{ITM}_0 - P^{OTM}_0)\times m - fee\\ & = & (61.58+0-29.08\times2-5.70-37.80+14.70\times2)\times100-1.00\times4\\ & = & -1072 \end{array}$$

So, the strategy losses $1,072.

The following algorithm implements a long put butterfly Option strategy:



Demonstration Algorithm
IndexOptionPutButterflyAlgorithm.py Python

Option Strategies

Call Calendar Spread

Introduction

Call calendar spread , also known as **call horizontal spread** , is a combination of a longer-term (far-leg/front-month) call and a shorter-term (near-leg/back-month) call, where all calls have the same underlying stock and the same strike price. The call calendar spread can be long or short.

Long Call Calendar Spread

The long call calendar spread consists of buying a longer-term call and selling a shorter-term call. This strategy profits from a decrease in the underlying price. It also profits from the time decay value because the theta θ (the Option price decay by 1 day closer to maturity) of the shorter-term call is larger than longer-term call.

Short Call Calendar Spread

The short call calendar spread consists of selling a longer-term call and buying a shorter-term call. The strategy profits from from an increase in the underlying price.

Implementation

Follow these steps to implement the call calendar spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 2, 19)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.Strikes(-1, 1).Expiration(timedelta(0), timedelta(62))
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```py
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the ATM strike
    atm_strike = sorted(chain, key=lambda x: abs(x.Strike - chain.Underlying.Price))[0].Strike

    # Select the ATM call Option contracts
    calls = [i for i in chain if i.Strike == atm_strike and i.Right == OptionRight.Call]
    if len(calls) == 0: return

    # Select the near and far expiry dates
    expiries = sorted([x.Expiry for x in calls])
    near_expiry = expiries[0]
    far_expiry = expiries[-1]
```

3. In the `OnData` method, call the `OptionStrategies.CallCalendarSpread` method and then submit the order.

```py
option_strategy = OptionStrategies.CallCalendarSpread(self.symbol, atm_strike, near_expiry,
far_expiry)
self.Buy(option_strategy, 1)    # if long call calendar spread
self.Sell(option_strategy, 1)   # if short call calendar spread
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` and `Sell` methods.

```py
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
self.Sell(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

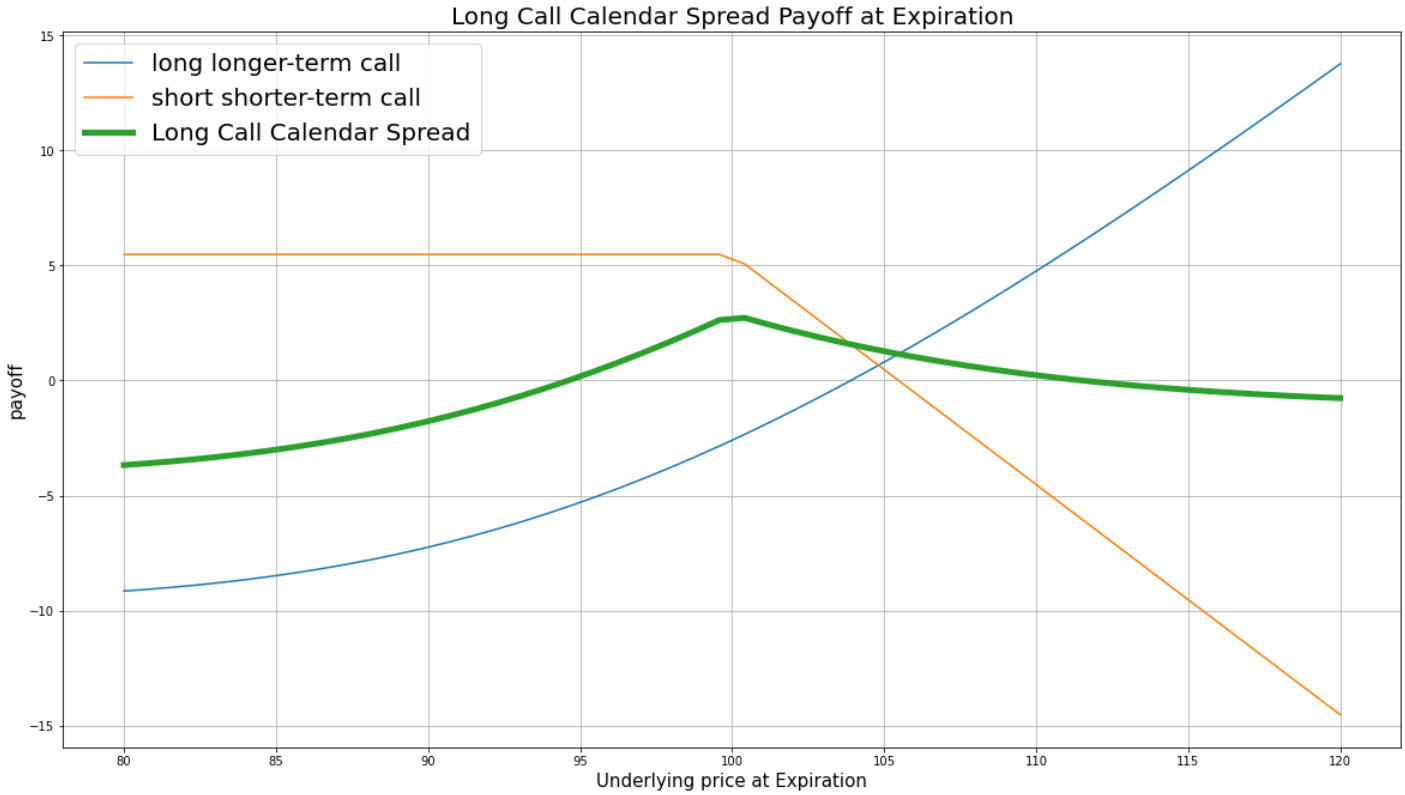
The call calendar spread can be long or short.

Long Call Calendar Spread

The long call calendar spread is a limited-reward-limited-risk strategy. The payoff at the shorter-term expiration is

$$\begin{array}{rcll} C^{\textrm{short-term}}_T & = & (S_T - K)^{+}\\ P_T & = & (C^{\textrm{long-term}}_T - C^{\textrm{short-term}}_T + C^{\textrm{short-term}}_0 - C^{\textrm{long-term}}_0)\times m - fee\\ \end{array}$$
$$\begin{array}{rcll} \textrm{where} & C^{\textrm{short-term}}_T & = & \textrm{Shorter term call value at time T}\\ & C^{\textrm{long-term}}_T & = & \textrm{Longer term call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{\textrm{short-term}}_0 & = & \textrm{Shorter term call value at position opening (credit received)}\\ & C^{\textrm{long-term}}_0 & = & \textrm{Longer term call value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of shorter term call expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread of the calls are at their maximum.

The maximum loss is the net debit paid, $C^{\textrm{short-term}}_0 - C^{\textrm{long-term}}_0$. It occurs when the underlying price moves very deep ITM or OTM so the values of both calls are close to zero.

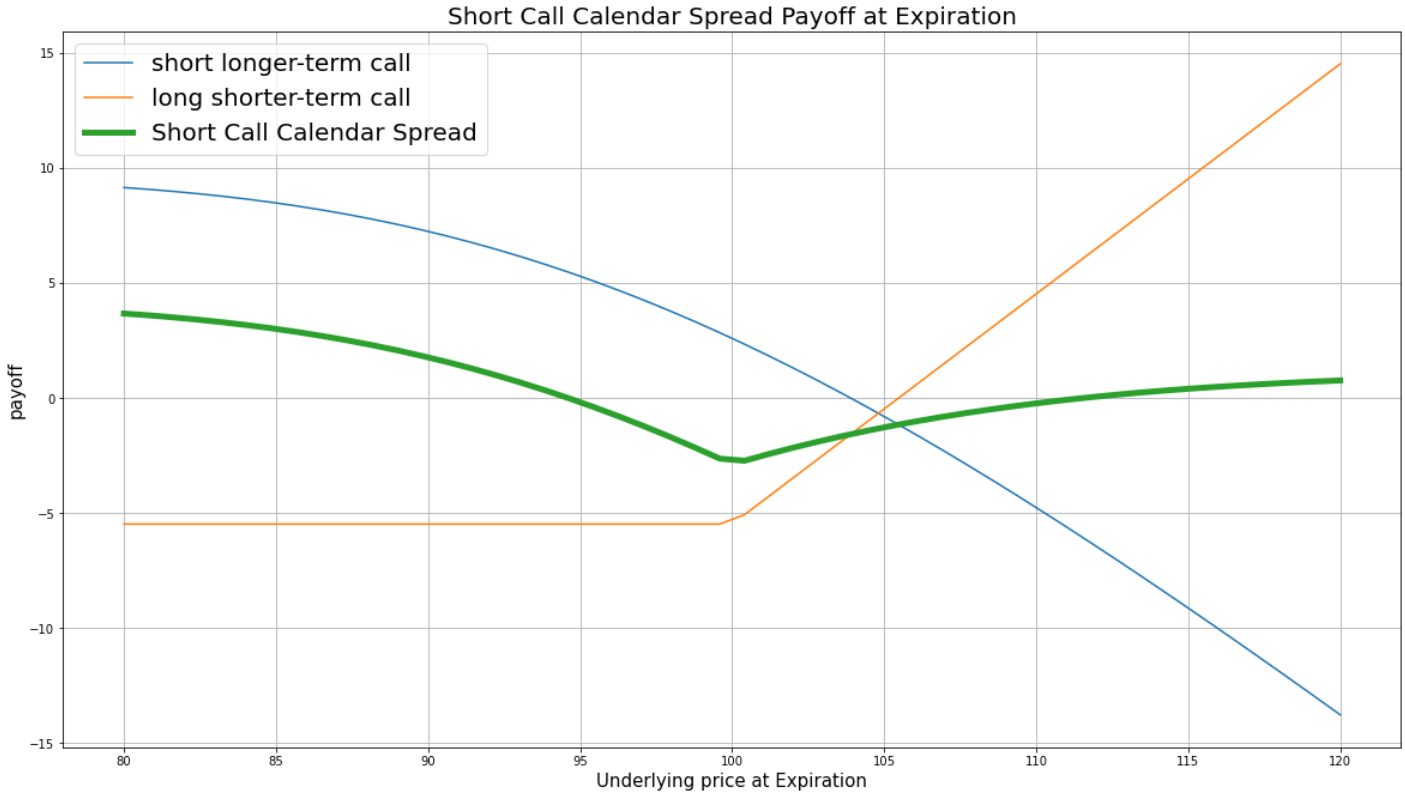
If the Option is American Option, there is risk of early assignment on the sold contract. If the buyer exercises the call you sell, you could lose all the debit you received if you don't close the long call and the underlying price drops below the long call strike price.

Short Call Calendar Spread

The short call calendar spread is a limited-reward-limited-risk strategy. The payoff at the shorter-term expiration is

$$\begin{array}{rcll} C^{\textrm{short-term}}_T & = & (S_T - K)^{+}\\ P_T & = & (C^{\textrm{short-term}}_T - C^{\textrm{long-term}}_T + C^{\textrm{long-term}}_0 - C^{\textrm{short-term}}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & C^{\textrm{short-term}}_T & = & \textrm{Shorter term call value at time T}\\ & C^{\textrm{long-term}}_T & = & \textrm{Longer term call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{\textrm{short-term}}_0 & = & \textrm{Shorter term call value at position opening (debit paid)}\\ & C^{\textrm{long-term}}_0 & = & \textrm{Longer term call value at position opening (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of shorter term call expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, C^{\textrm{long-term}}_0 - C^{\textrm{short-term}}_0. It occurs when the underlying price moves very deep ITM or OTM so the values of both calls are close to zero.

The maximum loss is undetermined because it depends on the underlying volatility. It occurs when S_T = S_0 and the spread of the 2 calls are at their maximum.

If the Option is American Option, there is risk of early assignment on the sold contract. If you don't close the call positions together, the naked short call will have unlimited drawdown risk after the long call expires.

Example

The following table shows the price details of the assets in the long version algorithm:

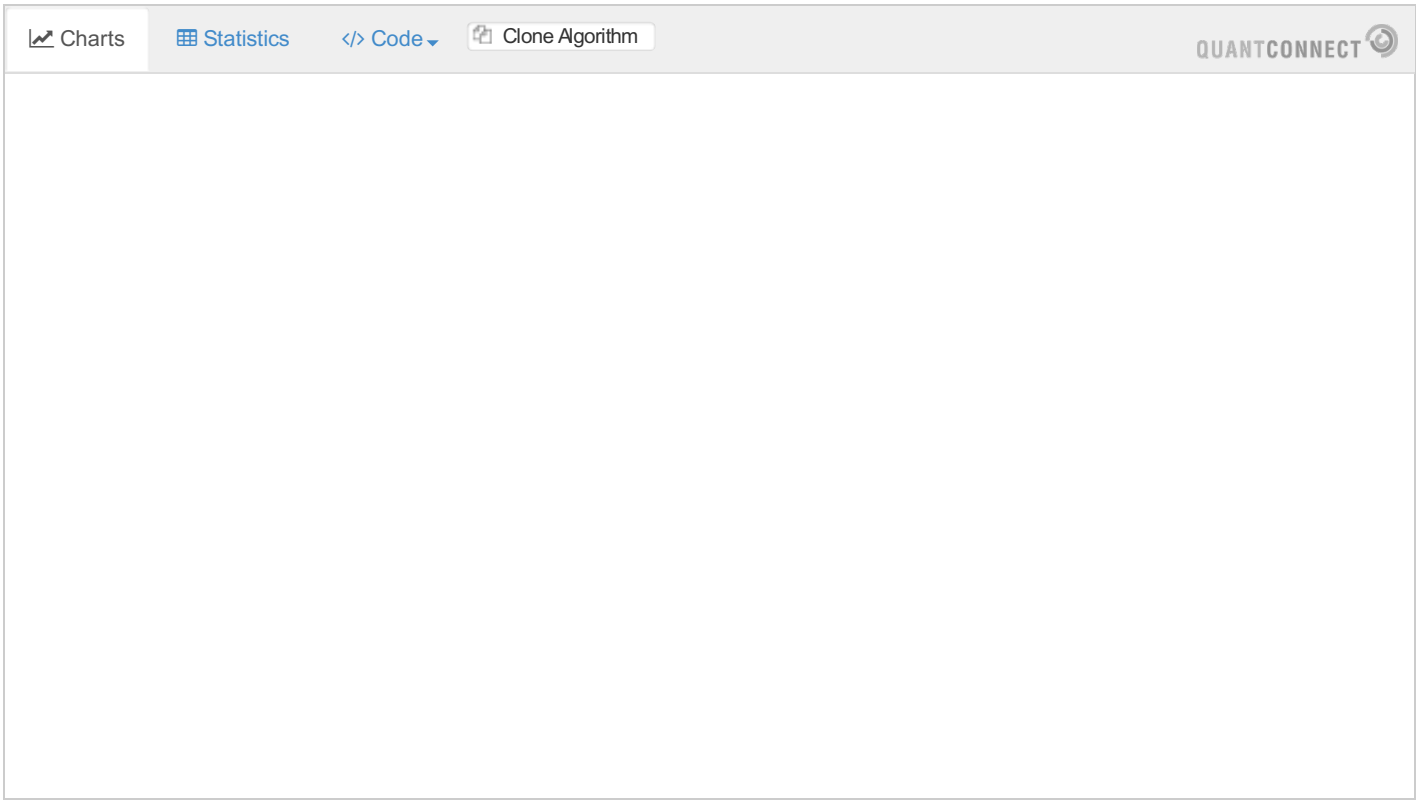
Asset	**Price ($)**	**Strike ($)**
Longer-term call at the start of the trade	4.40	835.00
Shorter-term call at the start of the trade	36.80	767.50
Longer-term call at time T	31.35	835.00
Underlying Equity at time T	829.08	-

Therefore, the payoff at time T (the expiration of the short-term call) is

\begin{array}{rcll} C^{\textrm{short-term}}_T & = & (S_T - K)^{+}\\ & = & (828.07-800.00)^{+}\\ & = & 28.07\\ P_T & = & (C^{\textrm{long-term}}_T - C^{\textrm{short-term}}_T + C^{\textrm{short-term}}_0 - C^{\textrm{long-term}}_0)\times m - fee\\ & = & (31.35-28.07+11.30-20.00)\times100-1.00\times2\\ & = & -544 \end{array}

So, the strategy losses $544.

The following algorithm implements a long call calendar spread Option strategy:



Demonstration Algorithm
IndexOptionCallCalendarSpreadAlgorithm.py Python

Option Strategies

Put Calendar Spread

Introduction

Put calendar spread , also known as **put horizontal spread** , is a combination of a longer-term (far-leg/front-month) put and a shorter-term (near-leg/back-month) put, where both puts have the same underlying stock and the same strike price. The put calendar spread strategy can be long or short.

Long Put Calendar Spread

The long put calendar spread consists of buying a longer-term put and selling a shorter-term put. This strategy profits from a decrease in price movement. The strategy also profits from the time decay value because the theta θ (the Option price decay by 1 day closer to maturity) of the shorter-term put is larger than the longer-term put.

Short Put Calendar Spread

The short put calendar spread consists of selling a longer-term put and buying a shorter-term put. This strategy profits from an increase in price movement.

Implementation

Follow these steps to implement the put calendar spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 2, 19)
    self.SetCash(500000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(self.UniverseFunc)

def UniverseFunc(self, universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return universe.Strikes(-1, 1).Expiration(timedelta(0), timedelta(62))
```

2. In the `OnData` method, select the strike price and expiration dates of the contracts in the strategy legs.

```py
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Get the ATM strike price
    atm_strike = sorted(chain, key=lambda x: abs(x.Strike - chain.Underlying.Price))[0].Strike

    # Select the ATM put contracts
    puts = [i for i in chain if i.Strike == atm_strike and i.Right == OptionRight.Put]
    if len(puts) == 0: return

    # Select the near and far expiration dates
    expiries = sorted([x.Expiry for x in puts], key = lambda x: x)
    near_expiry = expiries[0]
    far_expiry = expiries[-1]
```

3. In the `OnData` method, call the `OptionStrategies.PutCalendarSpread` method and then submit the order.

```py
option_strategy = OptionStrategies.PutCalendarSpread(self.symbol, atm_strike, near_expiry,
far_expiry)
self.Buy(option_strategy, 1)    # if long put calendar spread
self.Sell(option_strategy, 1)   # if short put calendar spread
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` and `Sell` methods.

```py
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
self.Sell(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

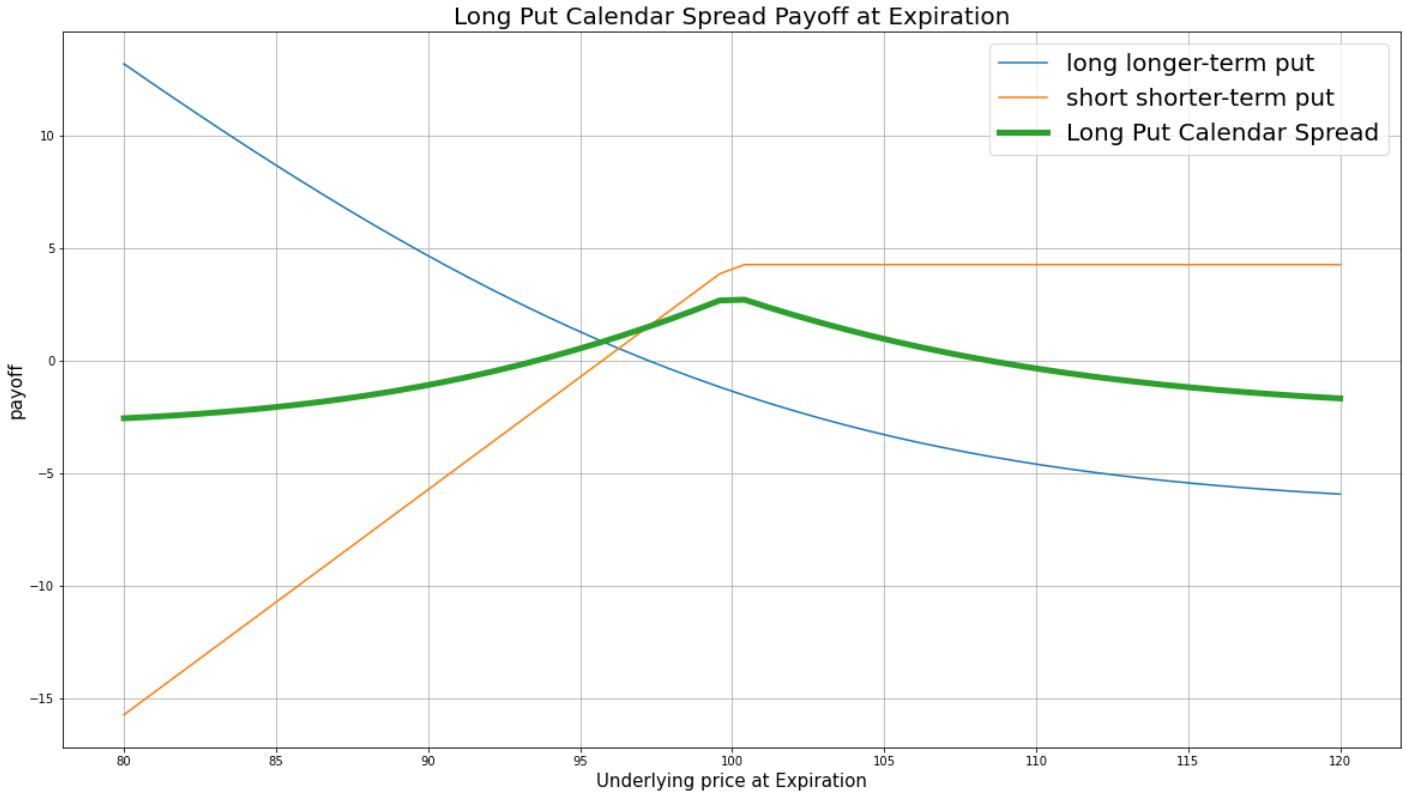
The put calendar spread can be long or short.

Long Put Calendar Spread

The long put calendar spread is a limited-reward-limited-risk strategy. The payoff is taken at the shorter-term expiration. The payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ P_T & = & (P^{\textrm{long-term}}_T - P^{\textrm{short-term}}_T + P^{\textrm{short-term}}_0 - P^{\textrm{long-term}}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & P^{\textrm{short-term}}_T & = & \textrm{Shorter term put value at time T}\\ & P^{\textrm{long-term}}_T & = & \textrm{Longer term put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & P^{\textrm{short-term}}_0 & = & \textrm{Shorter term put value at position opening (credit received)}\\ & P^{\textrm{long-term}}_0 & = & \textrm{Longer term put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of shorter term put expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread of the puts are at their maximum.

The maximum loss is the net debit paid, $P^{\textrm{short-term}}_0 - P^{\textrm{long-term}}_0$. It occurs when the underlying price moves very deep ITM or OTM so the values of both puts are close to zero.

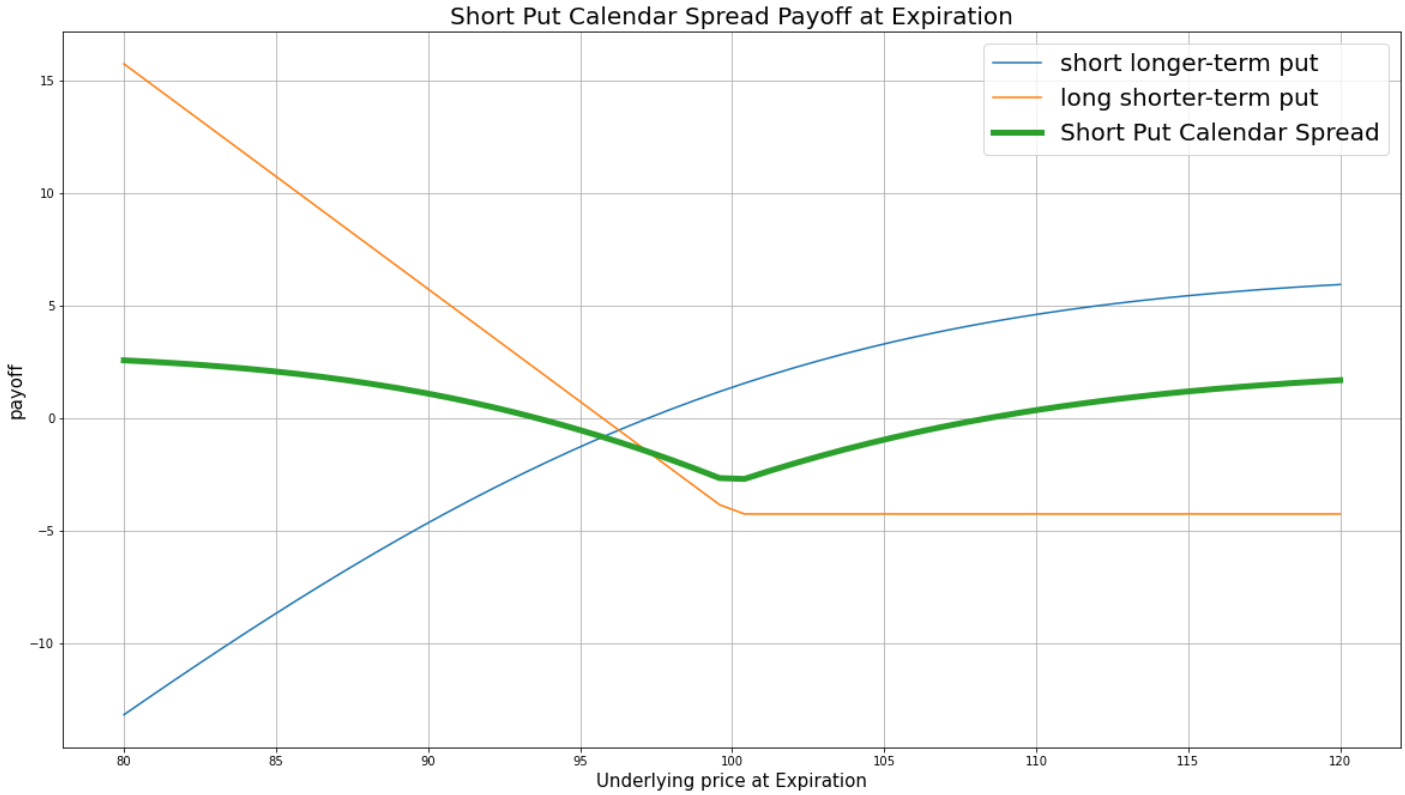
If the Option is American Option, there is risk of early assignment on the sold contract. Naked long puts pose risk of losing all the debit paid if you don't close the position with short put together and the price drops below its strike.

Short Put Calendar Spread

The short put calendar spread is a limited-reward-limited-risk strategy. The payoff is taken at the shorter-term expiration. The payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ P_T & = & (P^{\textrm{short-term}}_T - P^{\textrm{long-term}}_T + P^{\textrm{long-term}}_0 - P^{\textrm{short-term}}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & P^{\textrm{short-term}}_T & = & \textrm{Shorter term put value at time T}\\ & P^{\textrm{long-term}}_T & = & \textrm{Longer term put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & P^{\textrm{short-term}}_0 & = & \textrm{Shorter term put value at position opening (debit paid)}\\ & P^{\textrm{long-term}}_0 & = & \textrm{Longer term put value at position opening (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of shorter term put expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, $P^{\textrm{long-term}}_0 - P^{\textrm{short-term}}_0$. It occurs when the underlying price moves very deep ITM or OTM so the values of both puts are close to zero.

The maximum loss is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread of the 2 puts are at their maximum.

If the Option is American Option, there is risk of early assignment on the sold contract. Additionally, if you don't close the put positions together, the naked short put will have unlimited drawdown risk after the long put expires.

Example

The following table shows the price details of the assets in the long put calendar spread algorithm:

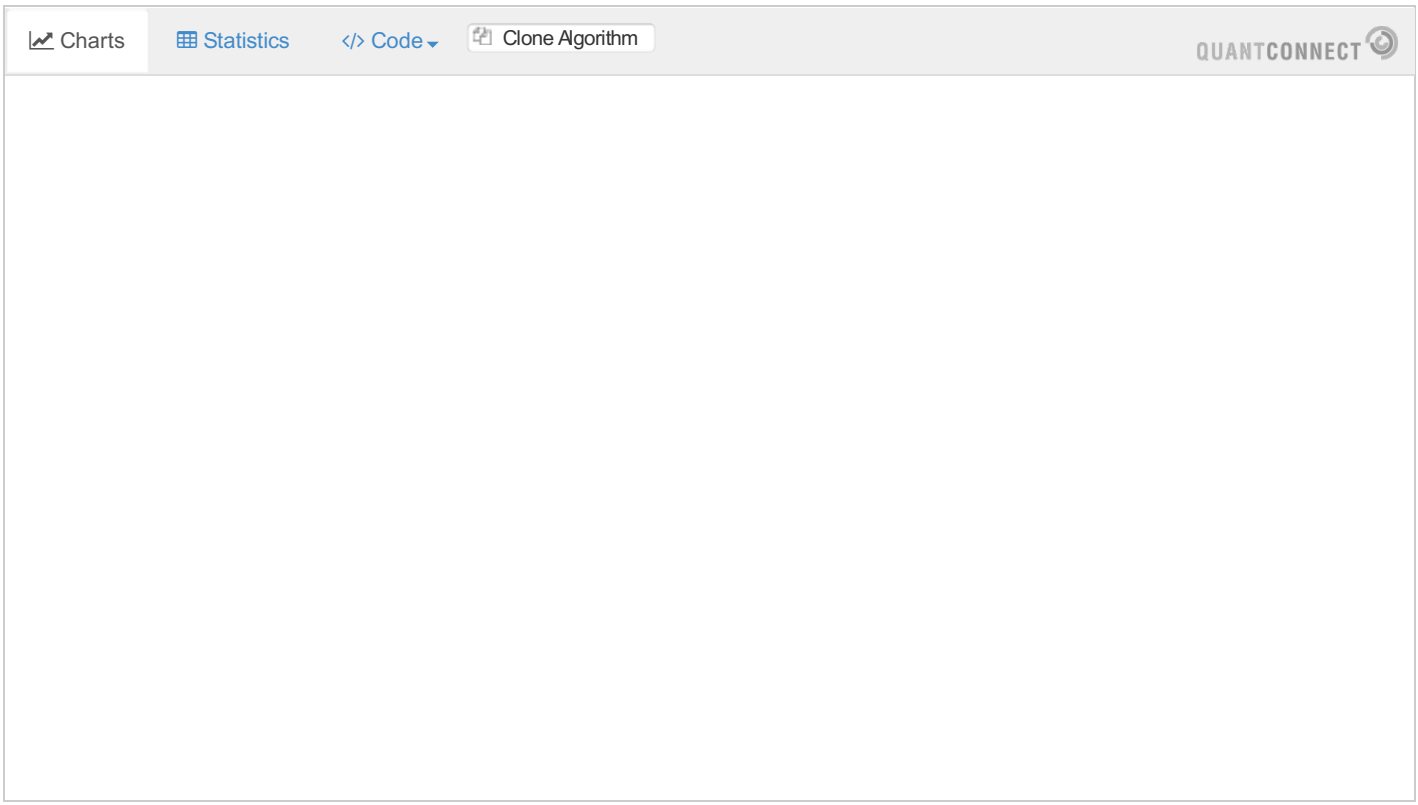
Asset	**Price ($)**	**Strike ($)**
Shorter-term put at position opening	11.30	800.00
Longer-term put at position opening	19.30	800.00
Longer-term put at shorter-term expiration	3.50	800.00
Underlying Equity at shorter-term expiration	828.07	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ & = & (800.00-828.07)^{+}\\ & = & 0\\ P_T & = & (P^{\textrm{long-term}}_T - P^{\textrm{short-term}}_T + P^{\textrm{short-term}}_0 - P^{\textrm{long-term}}_0)\times m - fee\\ & = & (3.50-0+11.30-19.30)\times100-1.00\times2\\ & = & -452\\ \end{array}$$

So, the strategy losses $452.

The following algorithm implements a long put calendar spread Option strategy:



Demonstration Algorithm
IndexOptionPutCalendarSpreadAlgorithm.py Python

Option Strategies

Covered Call

Introduction

A **Covered Call** consists of a long position in a stock and a short position in call Options for the same amount of stock. Covered calls aim to profit from the Option premium by selling calls written on the stock you already own. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, you keep the premium and still maintain the underlying Equity position. If the underlying price moves above the strike, the Option buyer can exercise the Options contract , which means you sell your stock at the strike price and keep the premium. Another risk of a covered call comes from the long stock position, which can drop in value.

Implementation

Follow these steps to implement the covered call strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2014, 1, 1)
    self.SetEndDate(2014, 3, 1)
    self.SetCash(100000)

    option = self.AddOption("IBM")
    self.symbol = option.Symbol
    self.call = None

    option.SetFilter(-3, 3, 0, 31)
```

2. In the `OnData` method, select the Option contract.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.call and self.Portfolio[self.call].Invested:
        return

    chain = slice.OptionChains.get(self.symbol)
    if not chain:
        return

    # Find ATM call with the farthest expiry
    expiry = max([x.Expiry for x in chain])
    call_contracts = sorted([x for x in chain
        if x.Right == OptionRight.Call and x.Expiry == expiry],
        key=lambda x: abs(chain.Underlying.Price - x.Strike))

    if not call_contracts:
        return

    atm_call = call_contracts[0]
```

3. In the `OnData` method, call the `OptionStrategies.CoveredCall` method and then submit the order.

```
covered_call = OptionStrategies.CoveredCall(self.symbol, atm_call.Strike, expiry)
self.Buy(covered_call, 1)

self.call = atm_call.Symbol
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

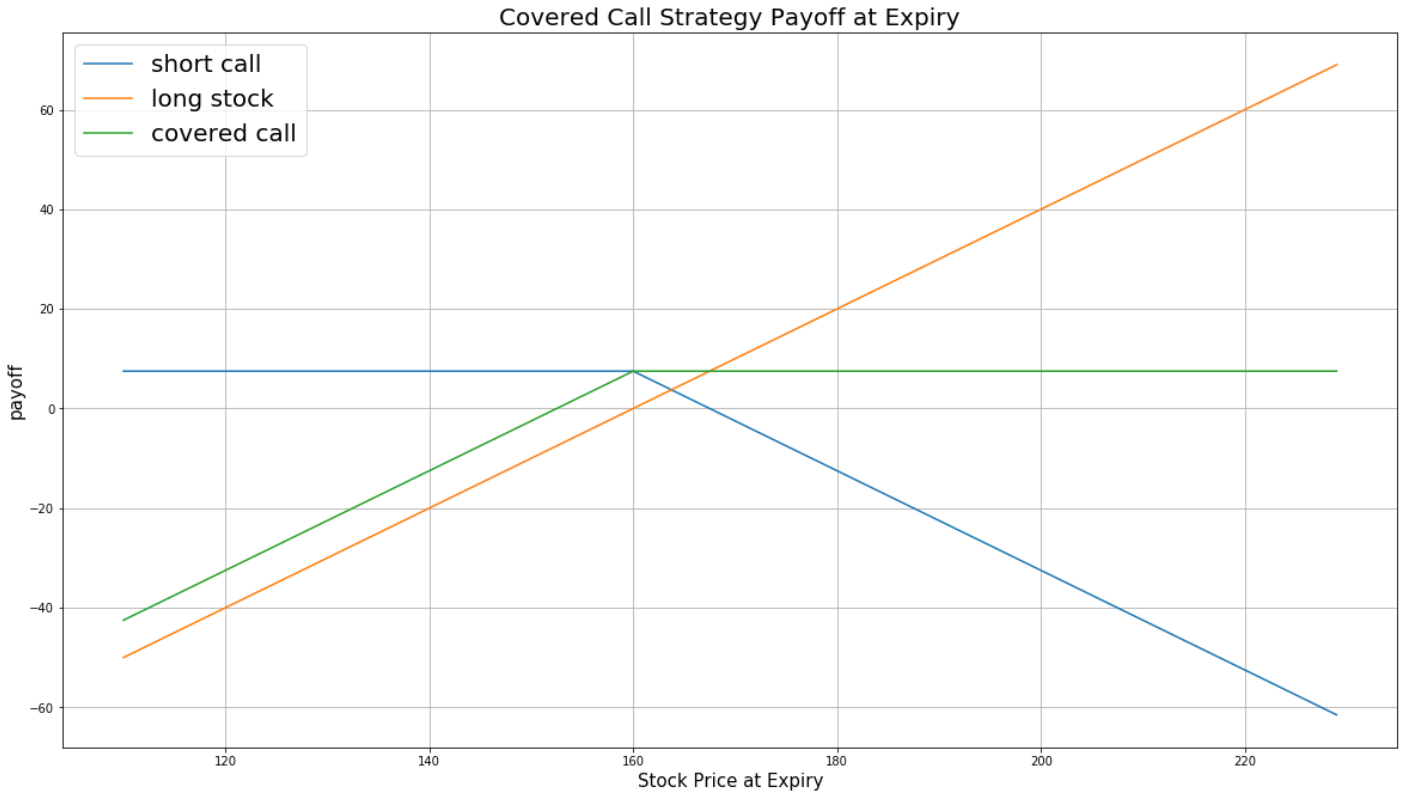
```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ P_T & = & (S_T - S_0 + C^{K}_0 - C^{K}_T)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & C^{K}_T & = & \textrm{Call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C^{K}_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K - S_T + C^{K}_0$, which occurs when the underlying price is at or above the strike price of the call at expiration.

The maximum loss is $S_0 - C^{K}_0$, which ocurrs when the underlying price drops.

If the Option is American Option, there is risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

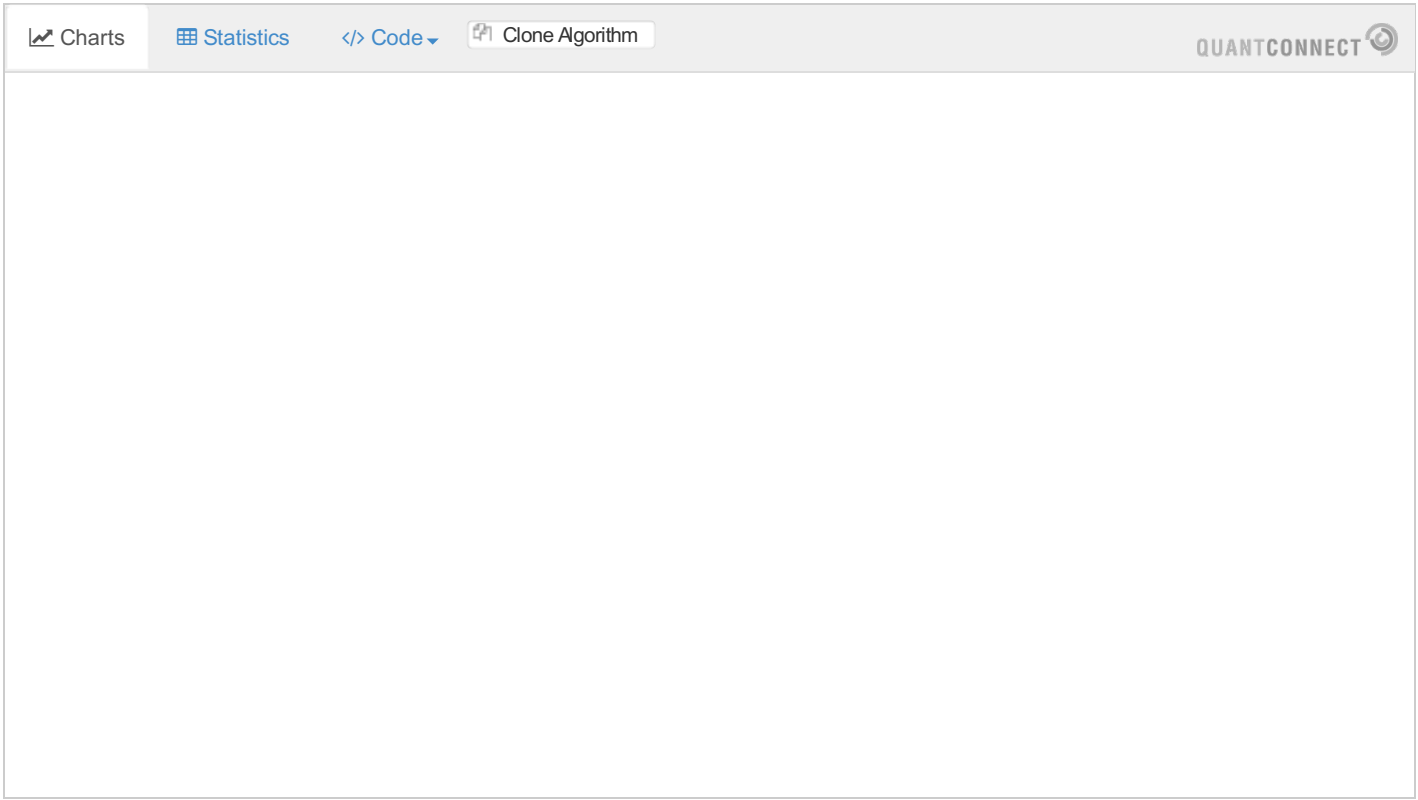
Asset	Price ($)	Strike ($)
Call	3.35	185.00
Underlying Equity at start of the trade	187.07	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ & = & (190.01 - 185)^{+}\\ & = & 5.01\\ P_T & = & (S_T - S_0 + C^{K}_0 - C^{K}_T)\times m - fee\\ & = & (190.01 - 187.07 + 3.35 - 5.01)\times m - fee\\ & = & 1.28 \times 100 - 2\\ & = & 126 \end{array}$$

So, the strategy gains $126.

The following algorithm implements a covered call strategy:



Option Strategies

Covered Put

Introduction

A **Covered Put** consists of a short position in a stock and a short position in put Options for the same amount of stock. Covered puts aim to profit from the Option premium by selling puts written on the stock you already shorted. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, you keep the premium and still maintain the underlying Equity position. If the underlying price moves above the strike, the Option buyer can exercise the Options contract , which mean you buy the stock at the strike price but you will still keep the premium. Another risk of a covered put comes from the short stock position, which can drop in value.

Implementation

Follow these steps to implement the covered put strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2014, 1, 1)
    self.SetEndDate(2014, 3, 1)
    self.SetCash(100000)

    option = self.AddOption("IBM")
    self.symbol = option.Symbol
    self.put = None

    option.SetFilter(-3, 3, 0, 31)
```

2. In the `OnData` method, select the Option contract.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.put and self.Portfolio[self.put].Invested:
        return

    chain = slice.OptionChains.get(self.symbol)
    if not chain:
        return

    # Find ATM put with the farthest expiry
    expiry = max([x.Expiry for x in chain])
    put_contracts = sorted([x for x in chain
        if x.Right == OptionRight.Put and x.Expiry == expiry],
        key=lambda x: abs(chain.Underlying.Price - x.Strike))

    if not put_contracts:
        return

    atm_put = put_contracts[0]
```

3. In the `OnData` method, put the `OptionStrategies.CoveredPut` method and then submit the order.

```
covered_put = OptionStrategies.CoveredPut(self.symbol, atm_put.Strike, expiry)
self.Buy(covered_put, 1)

self.put = atm_put.Symbol
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ P_T & = & (S_0 - S_T + P^{K}_0 - P^{K}_T)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & P^{K}_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & P^{K}_0 & = & \textrm{Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:

The maximum profit is $S_T - K + P^{K}_0$. It occurs when the underlying price is at or below the strike price of the put at expiration.

If the underlying price increase, the maximum loss is unlimited.

If the Option is American Option, there is risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

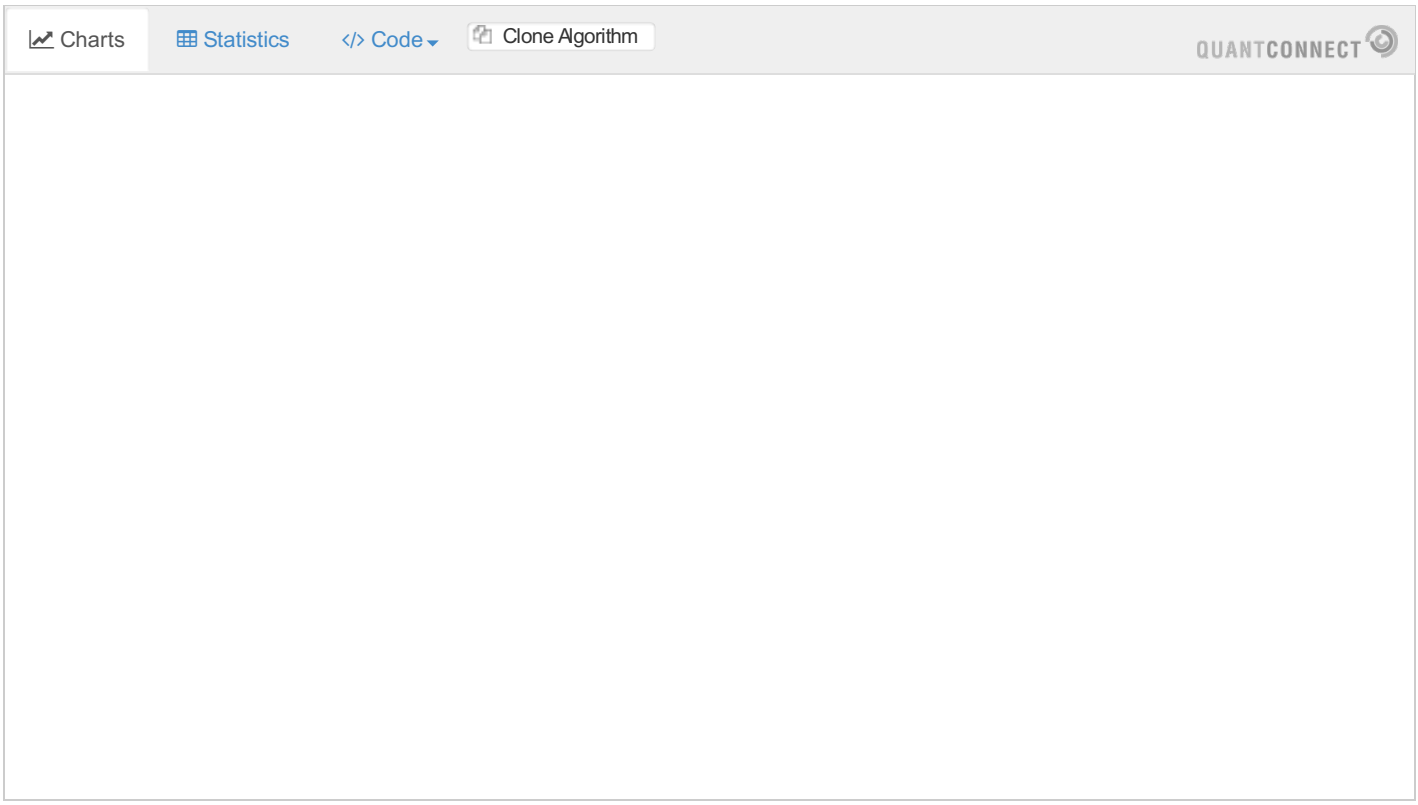
Asset	Price ($)	Strike ($)
Put	1.37	185.00
Underlying Equity at start of the trade	186.94	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ & = & (185 - 190.01)^{+}\\ & = & 0\\ P_T & = & (S_0 - S_T + P^{K}_0 - P^{K}_T)\times m - fee\\ & = & (186.94 - 190.01 + 1.37 - 0)\times m - fee\\ & = & -1.70 \times 100 - 2\\ & = & -172 \end{array}$$

So, the strategy loses $172.

The following algorithm implements a covered put strategy:


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Option Strategies

Iron Butterfly

Introduction

Warning: There is currently no `OptionStrategies` method for Iron Butterfly orders, so this tutorial manually orders the individual legs in the strategy. If you manually place multi-leg orders one at a time while there is no liquidity at a strike price, you can get stuck in an unhedged position.

The **Iron Butterfly** is an option strategy which involves four Option contracts. All the contracts have the same underlying stock and expiration, but the order of strike prices for the four contracts is A>B>C. The following table describes the strike price of each contract:

Position	Strike
1 OTM call	A
1 ATM call	B
1 ATM put	B
1 OTM put	C=B-(A-B)

The iron butterfly can be long or short.

Long Iron Butterfly

The long iron butterfly consists of selling an OTM call, selling an OTM put, buying an ATM call, and buying an ATM put. This strategy profits from a decrease in price movement (implied volatility).

Short Call Butterfly

The short call butterfly consists of buying an OTM call, buying an OTM put, selling an ATM call, and selling an ATM put. This strategy profits from an increase in price movement (implied volatility) and from time decay value since ATM options decay sharper.

Implementation

Follow these steps to implement the short iron butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2017, 4, 1)
    self.SetEndDate(2017, 5, 10)
    self.SetCash(100000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(-10, 10, timedelta(0), timedelta(30))
```

2. In the `OnData` method, select the contracts in the strategy legs.

```
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Separate the call and put contracts
    call = [i for i in chain if i.Right == 0]
    put = [i for i in chain if i.Right == 1]
    if len(call) == 0 or len(put) == 0 : return

    # Select the OTM contracts
    call_contracts = sorted(call, key = lambda x: x.Strike)
    put_contracts = sorted(put, key = lambda x: x.Strike)
    otm_call = call_contracts[-1]
    otm_put = put_contracts[0]

    # Select the ATM contracts
    atm_put = sorted(put_contracts, key = lambda x: abs(chain.Underlying.Price - x.Strike))[0]
    atm_call = sorted(call_contracts, key = lambda x: abs(chain.Underlying.Price - x.Strike))[0]
```

3. In the `OnData` method, submit the orders.

```
self.Sell(atm_put.Symbol, 1)
self.Sell(atm_call.Symbol, 1)
self.Buy(otm_call.Symbol, 1)
self.Buy(otm_put.Symbol, 1)
```

Strategy Payoff

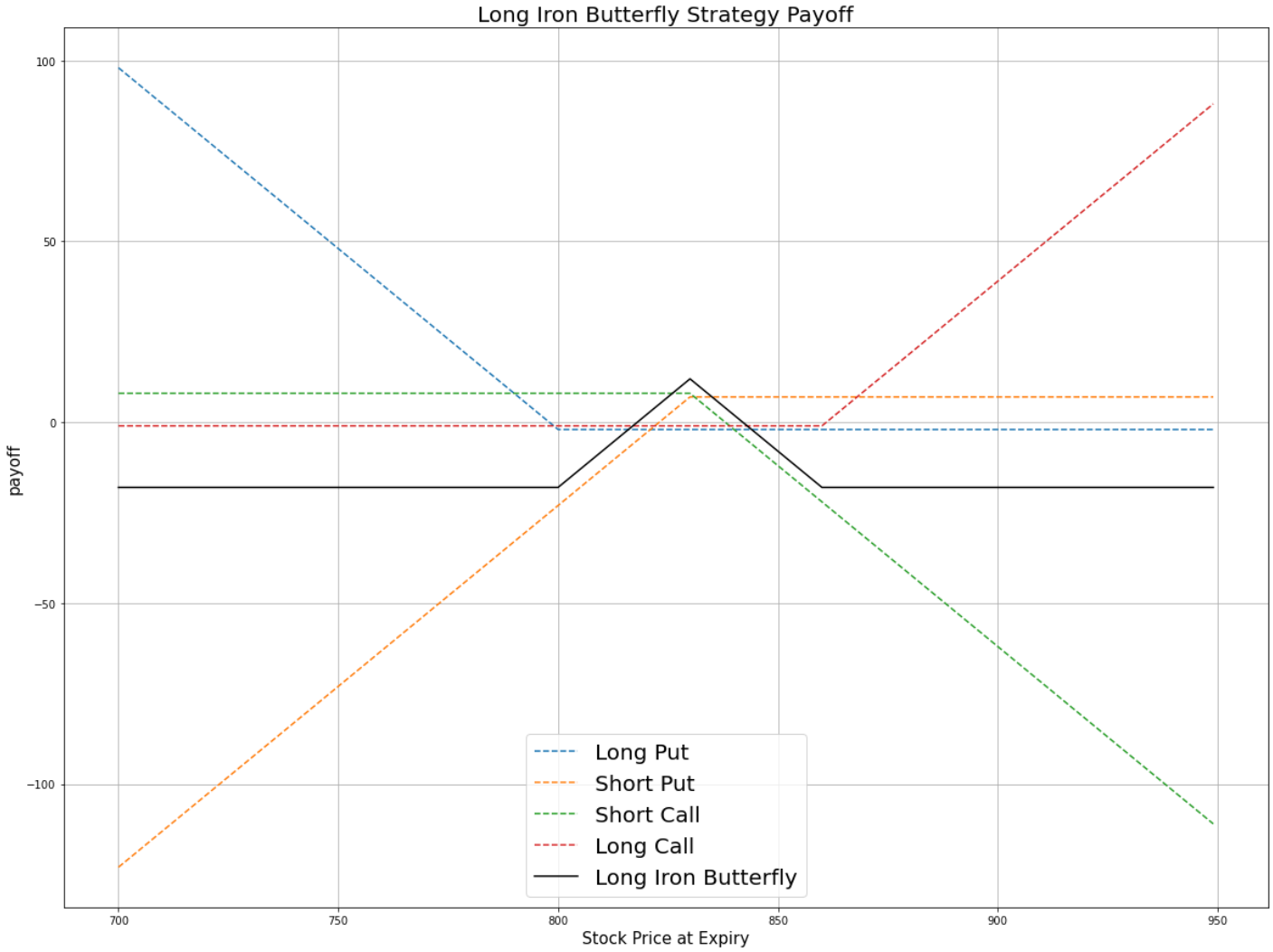
The iron butterfly can be long or short.

Long Iron Butterfly

The long iron butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{OTM}_T - P^{OTM}_T - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{OTM} & = & \textrm{OTM call strike price}\\ & K^C_{ATM} & = & \textrm{ATM call strike price}\\ & K^P_{OTM} & = & \textrm{OTM put strike price}\\ & K^P_{ATM} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (credit received)}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (debit paid)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (credit received)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is K^C_{OTM} - K^C_{ATM} - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0. It occurs when the underlying price is below the OTM put strike price or above the OTM call strike price at expiration.

The maximum loss is the net debit paid, C^{OTM}_0 + P^{OTM}_0 - C^{ATM}_0 - P^{ATM}_0. It occurs when the underlying price stays the same as when you opened the trade.

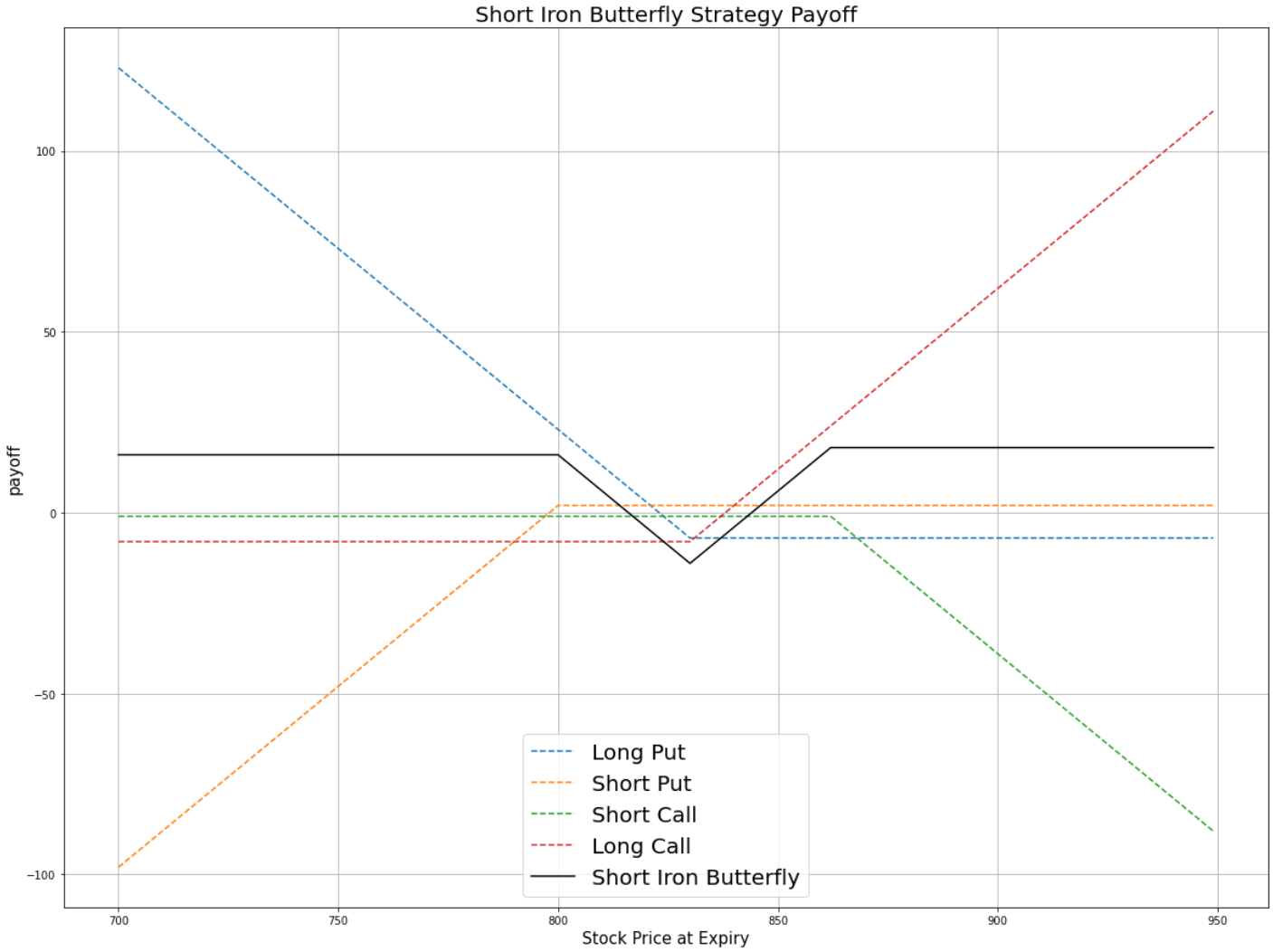
If the Option is American Option, there is a risk of early assignment on the sold contracts.

Short Call Butterfly

The short call butterfly is a limited-reward-limited-risk strategy. The payoff is

\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{ATM}_T - P^{ATM}_T - C^{OTM}_0 - P^{OTM}_0 + C^{ATM}_0 + P^{ATM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{OTM} & = & \textrm{OTM call strike price}\\ & K^C_{ATM} & = & \textrm{ATM call strike price}\\ & K^P_{OTM} & = & \textrm{OTM put strike price}\\ & K^P_{ATM} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (debit paid)}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (credit received)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (debit paid)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, C^{ATM}_0 + P^{ATM}_0 - C^{OTM}_0 - P^{OTM}_0. It occurs when the underlying price stays the same as when you opened the trade.

The maximum loss is K^C_{OTM} - K^C_{ATM} - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0. It occurs when the underlying price is below the OTM put strike price or above the OTM call strike price at expiration.

If the Option is American Option, there is a risk of early assignment on the sold contracts.

Example

The following table shows the price details of the assets in the algorithm:

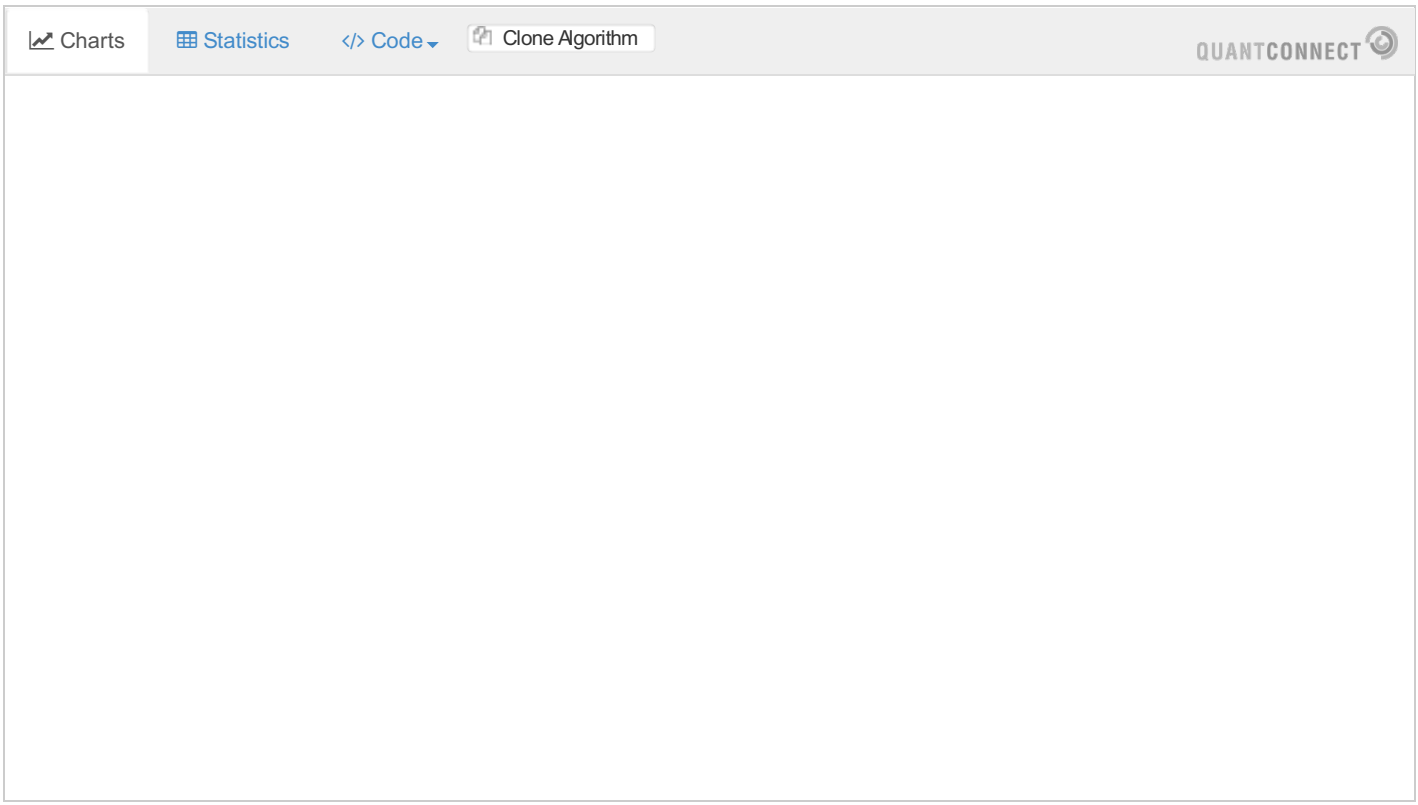
Asset	**Price ($)**	**Strike ($)**
OTM call	1.85	857.50
OTM put	2.75	810.00
ATM call	8.10	832.00
ATM put	7.40	832.00
Underlying Equity at expiration	851.20	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ & = & (851.20-857.50)^{+}\\ & = & 0\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ & = & (851.20-832.00)^{+}\\ & = & 19.20\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ & = & (832.00-851.20)^{+}\\ & = & 0\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ & = & (810.00-851.20)^{+}\\ & = & 0\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{ATM}_T - P^{ATM}_T - C^{OTM}_0 - P^{OTM}_0 + C^{ATM}_0 + P^{ATM}_0)\times m - fee\\ & = & (0+0-19.20-0+8.10+7.40-1.85-2.75)\times100-1\times4\\ & = & -834 \end{array}$$

So, the strategy losses $834.

The following algorithm implements a short iron butterfly Option strategy:



Option Strategies

Iron Condor

Introduction

The **Iron Condor** is an Option strategy that consists of four contracts. All the contracts have the same underlying Equity and expiration, but the order of strike prices is A>B>C>D. The following table describes the strike prices of each contract:

Position	Strike
1 far-OTM call	A
1 near-OTM call	B, where B > underlying\ price
1 near-OTM put	C, where C < underlying\ price
1 far-OTM put	D, where C-D = A-B

The iron condor can be long or short.

Long Iron Condor

The long iron condor consists of selling a far OTM call, selling a far OTM put, buying a near OTM call, and buying a near OTM put. This strategy profits from a increase in price movement (implied volatility).

Short Iron Condor

The short iron condor consists of buying a far OTM call, buying a far OTM put, selling a near ATM call, and selling a near ATM put. This strategy profits from an decrease in price movement (implied volatility) and time decay since ATM options decay sharper.

Implementation

Follow these steps to implement the long iron condor strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2017, 2, 1)
    self.SetEndDate(2017, 3, 1)
    self.SetCash(500000)

    option = self.AddOption("GOOG")
    self.symbol = option.Symbol
    option.SetFilter(lambda universe:
        universe.IncludeWeeklys().Strikes(-15, 15).Expiration(0, 40))
```

2. In the `OnData` method, select the contracts in the strategy legs.

```
def OnData(self, slice: Slice) -> None:
    if self.Portfolio[self.symbol.Underlying].Invested:
        self.Liquidate()

    if self.Portfolio.Invested or not self.IsMarketOpen(self.symbol):
        return

    chain = slice.OptionChains.get(self.symbol)
    if not chain:
        return

    # Find put and call contracts with the farthest expiry
    expiry = max([x.Expiry for x in chain])
    chain = sorted([x for x in chain if x.Expiry == expiry], key = lambda x: x.Strike)

    put_contracts = [x for x in chain if x.Right == OptionRight.Put]
    call_contracts = [x for x in chain if x.Right == OptionRight.Call]

    if len(call_contracts) < 10 or len(put_contracts) < 10:
        return

    # Select the strikes in the strategy legs
    far_put = put_contracts[0].Strike
    near_put = put_contracts[10].Strike
    near_call = call_contracts[-10].Strike
    far_call = call_contracts[-1].Strike
```

3. In the `OnData` method, call the `OptionStrategies.IronCondor` method and then submit the order.

```
iron_condor = OptionStrategies.IronCondor(
    self.symbol,
    far_put,
    near_put,
    near_call,
    far_call,
    expiry)

self.Buy(iron_condor, 2)
```

Strategy Payoff

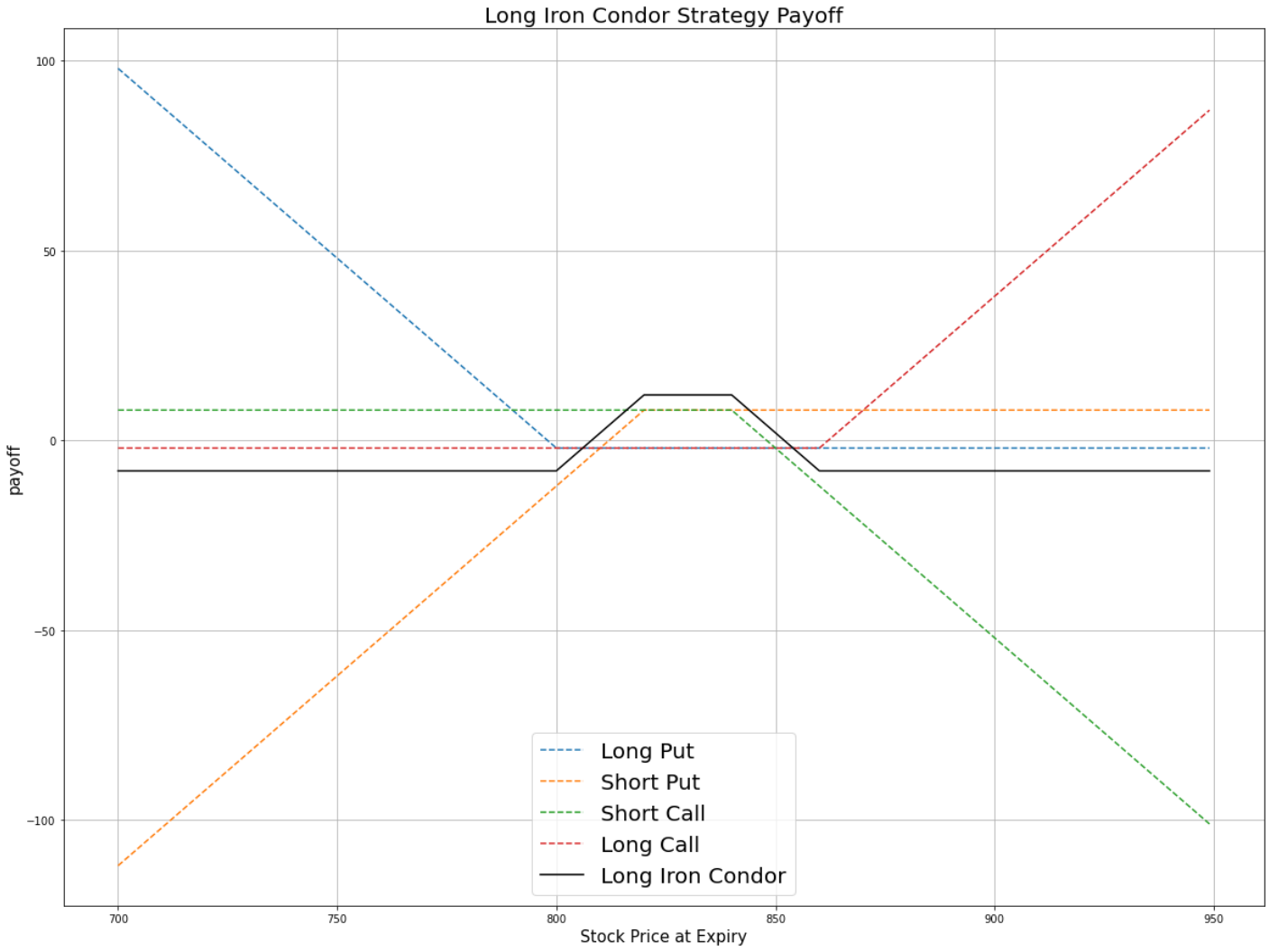
The iron condor can be long or short.

Long Iron Condor

This is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ P_T & = & (C^{near}_T + P^{near}_T - C^{far}_T - P^{far}_T - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & & & \\ C^{far}_T & = & \textrm{Far OTM call value at time T}\\ C^{near}_T & = & \textrm{Near OTM call value at time T}\\ P^{far}_T & = & \textrm{Far OTM put value at time T}\\ P^{near}_T & = & \textrm{Near ATM put value at time T}\\ S_T & = & \textrm{Underlying asset price at time T}\\ K^C_{far} & = & \textrm{Far OTM call strike price}\\ K^C_{near} & = & \textrm{Near OTM call strike price}\\ K^P_{far} & = & \textrm{Far OTM put strike price}\\ K^P_{near} & = & \textrm{Near OTM put strike price}\\ P_T & = & \textrm{Payout total at time T}\\ C^{far}_0 & = & \textrm{Far OTM call value at position opening (credit received)}\\ C^{near}_0 & = & \textrm{Near OTM call value at position opening (debit paid)}\\ P^{far}_0 & = & \textrm{Far OTM put value at position opening (credit received)}\\ P^{near}_0 & = & \textrm{Near OTM put value at position opening (debit paid)}\\ m & = & \textrm{Contract multiplier}\\ T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is K^C_{far} - K^C_{near} - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0, where K^P_{OTM} > S_T or S_T > K^C_{OTM}.

The maximum loss is the net debit paid: C^{far}_0 + P^{far}_0 - C^{near}_0 - P^{near}_0, where K^P_{OTM} < S_T < K^C_{OTM}.

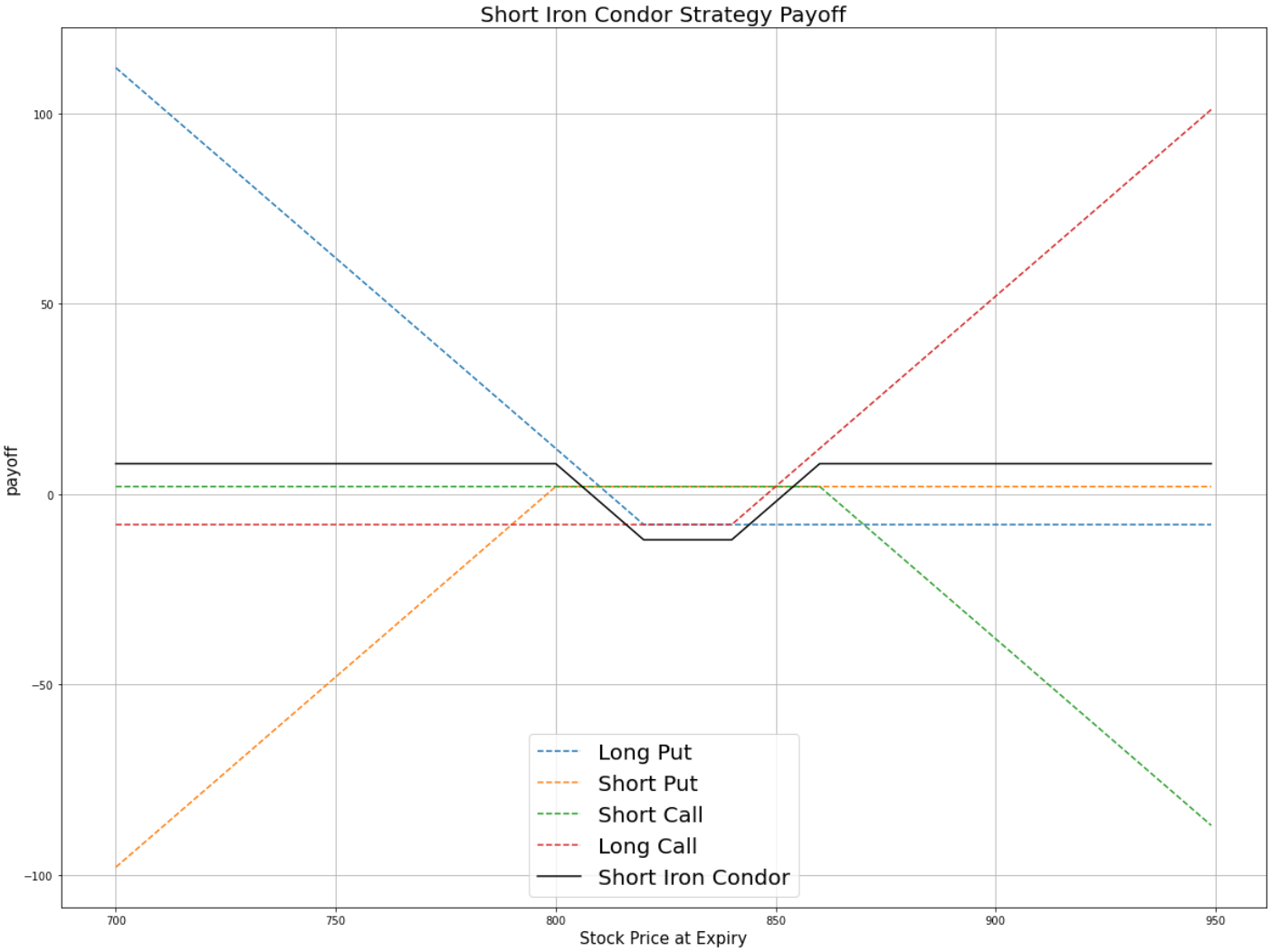
If the Option is American Option, there is a risk of early assignment on the sold contracts.

Short Iron Condor

This is a limited-reward-limited-risk strategy. The payoff is

\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ P_T & = & (C^{far}_T + P^{far}_T - C^{near}_T - P^{near}_T - C^{far}_0 - P^{far}_0 + C^{near}_0 + P^{near}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{far}_T & = & \textrm{Far OTM call value at time T}\\ & C^{near}_T & = & \textrm{Near OTM call value at time T}\\ & P^{far}_T & = & \textrm{Far OTM put value at time T}\\ & P^{near}_T & = & \textrm{Near ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{far} & = & \textrm{Far OTM call strike price}\\ & K^C_{near} & = & \textrm{Near OTM call strike price}\\ & K^P_{far} & = & \textrm{Far OTM put strike price}\\ & K^P_{near} & = & \textrm{Near OTM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{far}_0 & = & \textrm{Far OTM call value at position opening (credit received)}\\ & C^{near}_0 & = & \textrm{Near OTM call value at position opening (debit paid)}\\ & P^{far}_0 & = & \textrm{Far OTM put value at position opening (credit received)}\\ & P^{near}_0 & = & \textrm{Near OTM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}

The following chart shows the payoff at expiration:



The maximum profit is the net credit received after commission when opening the trade, where K^P_{OTM} < S_T < K^C_{OTM}.

The maximum loss is K^C_{far} - K^C_{near} + C^{near}_0 + P^{near}_0 - C^{far}_0 - P^{far}_0, where K^P_{OTM} > S_T or S_T > K^C_{OTM}.

If the Option is American Option, there is risk of early assignment on the sold contracts.

Example

The following table shows the price details of the assets in the algorithm:

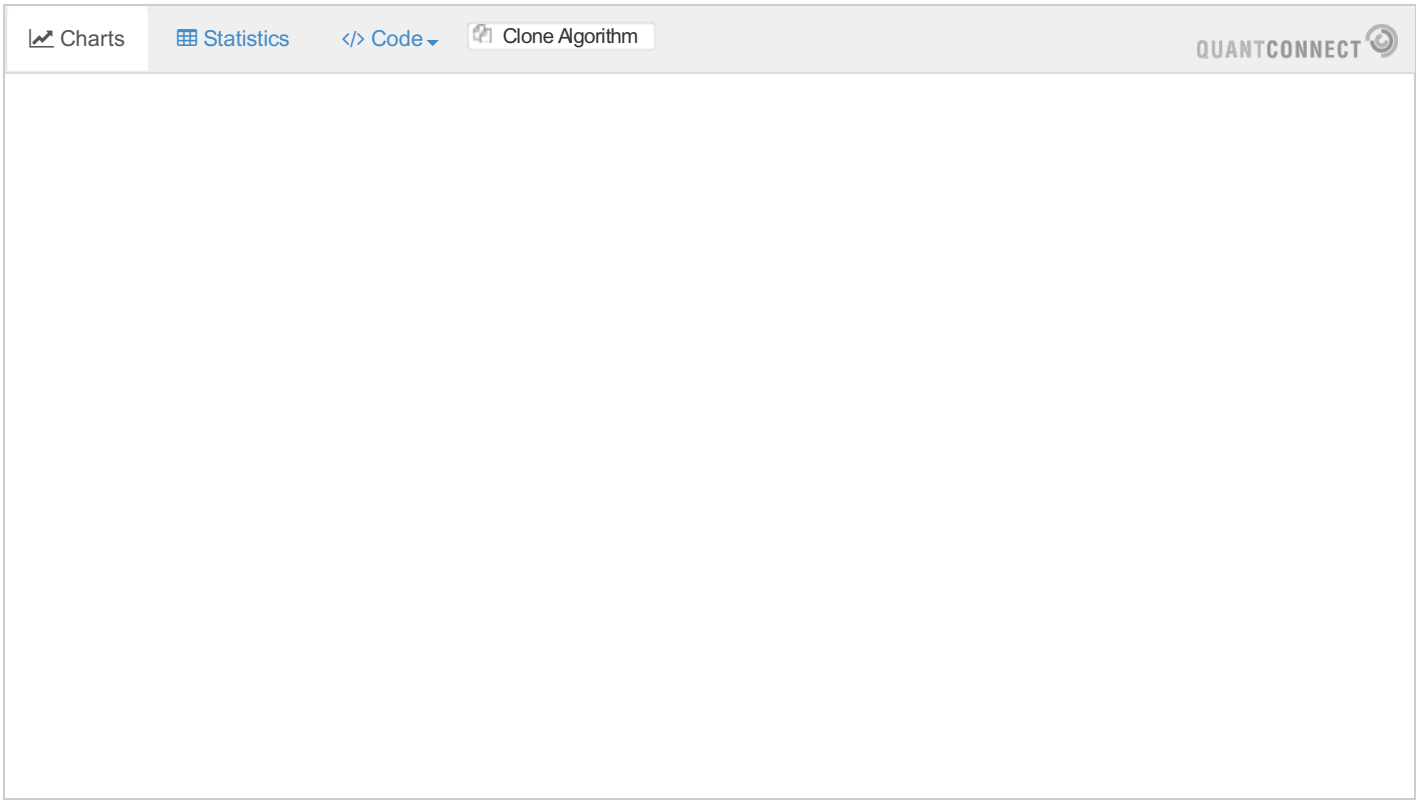
Asset	**Price ($)**	**Strike ($)**
Far-OTM call	1.85	857.50
Far-OTM put	2.75	810.00
Near-OTM call	1.35	855.00
Near-OTM put	2.15	815.00
Underlying Equity at expiration	851.20	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ & = & (851.20-857.50)^{+}\\ & = & 0\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ & = & (851.20-855.00)^{+}\\ & = & 0\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ & = & (815.00-851.20)^{+}\\ & = & 0\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ & = & (810.00-851.20)^{+}\\ & = & 0\\ P_T & = & (C^{near}_T + P^{near}_T - C^{far}_T - P^{far}_T - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0)\times m - fee\\ & = & (0+0-0-0+1.35+2.15-1.85-2.75)\times100-1\times4\\ & = & -114 \end{array}$$

So, the strategy losses $114.

The following algorithm implements a long iron condor Option strategy:



Demonstration Algorithm
IndexOptionIronCondorAlgorithm.py Python

Option Strategies

Protective Call

Introduction

A **Protective Call** consists of a short position in a stock and a long position in a call Option for the same amount of stock. Protective calls aim to hedge the short position of a stock with a long ATM or slightly OTM call Option. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, the Option contract becomes worthless but the short position acquires an unrealized gain. If the underlying price moves above the strike, you can exercise the Options contract and receive the underlying Equity, which closes your short position.

Implementation

Follow these steps to implement the protective call strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Options universe .

```
def Initialize(self) -> None:
    self.SetStartDate(2014, 1, 1)
    self.SetEndDate(2014, 3, 1)
    self.SetCash(100000)

    option = self.AddOption("IBM")
    self.symbol = option.Symbol
    self.call = None

    option.SetFilter(-3, 3, 0, 31)
```

2. In the `OnData` method, select the Option contract.

```
def OnData(self, slice: Slice) -> None:
    if self.call and self.Portfolio[self.call].Invested:
        return

    chain = slice.OptionChains.get(self.symbol)
    if not chain:
        return

    # Find ATM call with the farthest expiry
    expiry = max([x.Expiry for x in chain])
    call_contracts = sorted([x for x in chain
        if x.Right == OptionRight.Call and x.Expiry == expiry],
        key=lambda x: abs(chain.Underlying.Price - x.Strike))

    if not call_contracts:
        return

    atm_call = call_contracts[0]
```

3. In the `OnData` method, call the `OptionStrategies.ProtectiveCall` method and then submit the order.

```
protective_call = OptionStrategies.ProtectiveCall(self.symbol, atm_call.Strike, expiry)
self.Buy(protective_call, 1)

self.call = atm_call.Symbol
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False`. You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ P_T & = & (S_0 - S_T + C^{K}_T - C^{K}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & C^{K}_T & = & \textrm{Call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C^{K}_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:

The maximum profit is $S_0 - C^{K}_0$, which occurs when the underlying price is 0.

The maximum loss is $S_0 - K - C^{K}_0$, which occurs when the underlying price is above the strike price.

Example

The following table shows the price details of the assets in the algorithm:

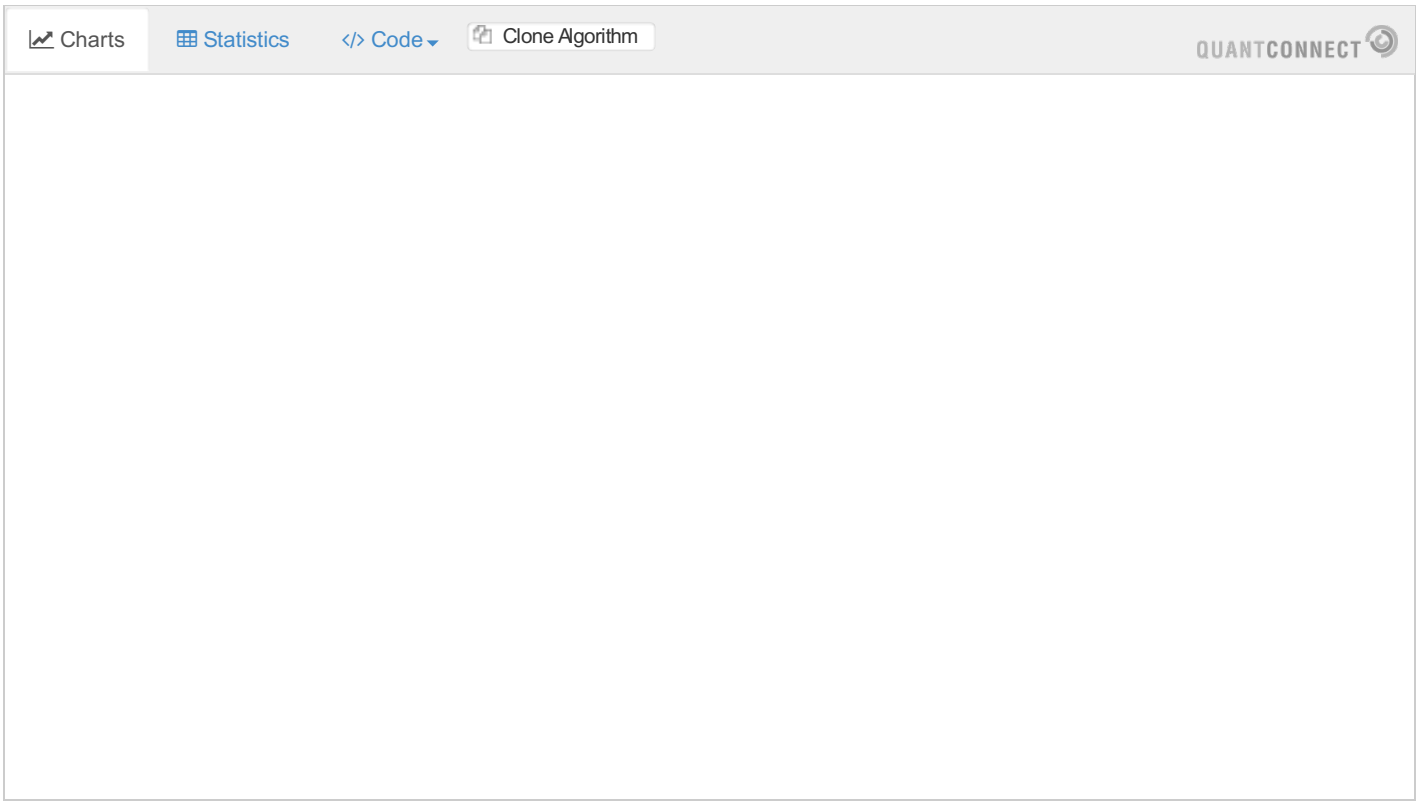
Asset	Price ($)	Strike ($)
Call	3.50	185.00
Underlying Equity at start of the trade	186.94	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ & = & (190.01 - 185)^{+}\\ & = & 5.01\\ P_T & = & (S_0 - S_T + C^{K}_T - C^{K}_0)\times m - fee\\ & = & (186.94 - 190.01 + 5.01 - 3.50)\times m - fee\\ & = & -1.56 \times 100 - 2\\ & = & -158 \end{array}$$

So, the strategy losses $158.

The following algorithm implements a protective call Option strategy:


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Option Strategies

Protective Put

Introduction

A **Protective Put** consists of buying a long position in a stock and a long position in put Options for the same amount of stock. Protective puts aim to hedge the long position of a stock with a long ATM or slightly OTM put Option. At any time for American Options or at expiration for European Options, if the stock moves above the strike price, the Option contract becomes worthless but the long stock position acquires an unrealized gain. If the underlying price moves below the strike, you can exercise the Options contract , which sells your underlying position at the put Option strike price.

Implementation

Follow these steps to implement the protective put strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2014, 1, 1)
    self.SetEndDate(2014, 3, 1)
    self.SetCash(100000)

    option = self.AddOption("IBM")
    self.symbol = option.Symbol
    self.put = None

    option.SetFilter(-3, 3, 0, 31)
```

2. In the `OnData` method, select the Option contract.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.put and self.Portfolio[self.put].Invested:
        return

    chain = slice.OptionChains.get(self.symbol)
    if not chain:
        return

    # Find ATM put with the farthest expiry
    expiry = max([x.Expiry for x in chain])
    put_contracts = sorted([x for x in chain
        if x.Right == OptionRight.Put and x.Expiry == expiry],
        key=lambda x: abs(chain.Underlying.Price - x.Strike))

    if not put_contracts:
        return

    atm_put = put_contracts[0]
```

3. In the `OnData` method, call the `OptionStrategies.ProtectivePut` method and then submit the order.

```
protective_put = OptionStrategies.ProtectivePut(self.symbol, atm_put.Strike, expiry)
self.Buy(protective_put, 1)

self.put = atm_put.Symbol
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ P_T & = & (S_T - S_0 + P^{K}_T - P^{K}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & P^{K}_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & P^{K}_0 & = & \textrm{Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:

The maximum profit is $S_T - S_0 - P^{K}_0$, which occurs when the underlying price is above the $S_0 + P^{K}_0$.

The maximum loss is P^{K}_0, which occurs when the underlying price drops.

Example

The following table shows the price details of the assets in the algorithm:

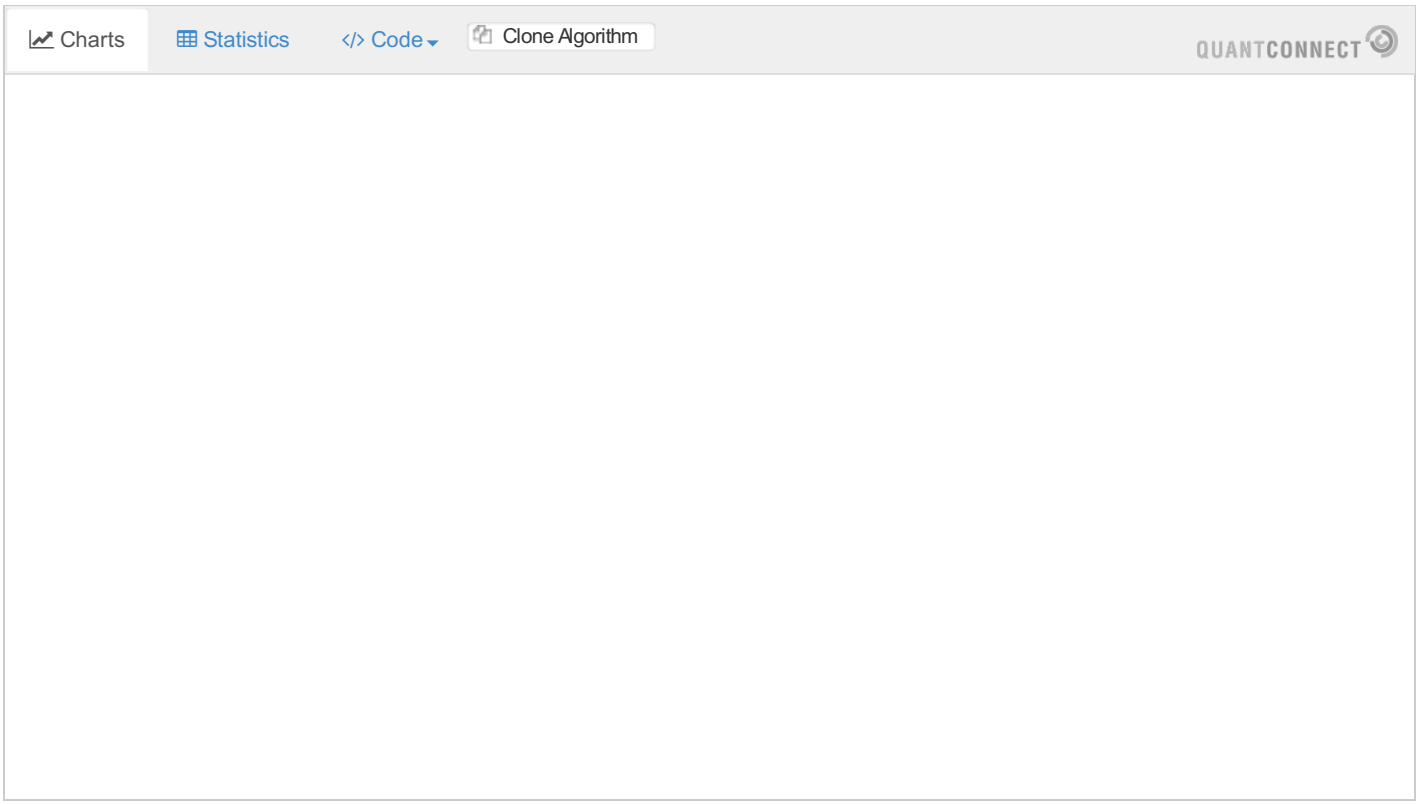
Asset	Price ($)	Strike ($)
Put	1.53	185.00
Underlying Equity at start of the trade	187.07	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ & = & (185 - 190.1)^{+}\\ & = & 0\\ P_T & = & (S_T - S_0 + P^{K}_T - P^{K}_0)\times m - fee\\ & = & (190.01 - 187.07 + 0 - 1.53)\times m - fee\\ & = & 1.41 \times 100 - 2\\ & = & 139 \end{array}$$

So, the strategy gains $139.

The following algorithm implements a protective put Option strategy:


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Option Strategies

Protective Collar

Introduction

A **Protective Collar** is an Options strategy that consists of a covered call and a long put (protective put) with a lower strike price than the short call contract. In contrast to the covered call, the protective put component limits the drawdown of the strategy when the underlying price decreases too much.

Implementation

Follow these steps to implement the protective collar strategy:

1. In the `Initialize` method, set the start date, set the end date, subscribe to the underlying Equity , and create an Option universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2017, 4, 1)
    self.SetEndDate(2017, 4, 30)
    self.SetCash(100000)

    self.equity_symbol = self.AddEquity("GOOG").Symbol
    option = self.AddOption("GOOG", Resolution.Minute)
    self.option_symbol = option.Symbol
    option.SetFilter(-10, +10, timedelta(0), timedelta(30))
```

2. In the `OnData` method, select the Option contracts.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Select an expiry date
    expiry = sorted(chain, key = lambda x: x.Expiry)[-1].Expiry

    # Select the call and put contracts that expire on the selected date
    calls = [x for x in chain if x.Right == OptionRight.Call and x.Expiry == expiry]
    puts = [x for x in chain if x.Right == OptionRight.Put and x.Expiry == expiry]
    if not calls or not puts: return

    # Select the OTM contracts
    call = sorted(calls, key = lambda x: x.Strike)[-1]
    put = sorted(puts, key = lambda x: x.Strike)[0]
```

3. In the `OnData` method, submit the orders.

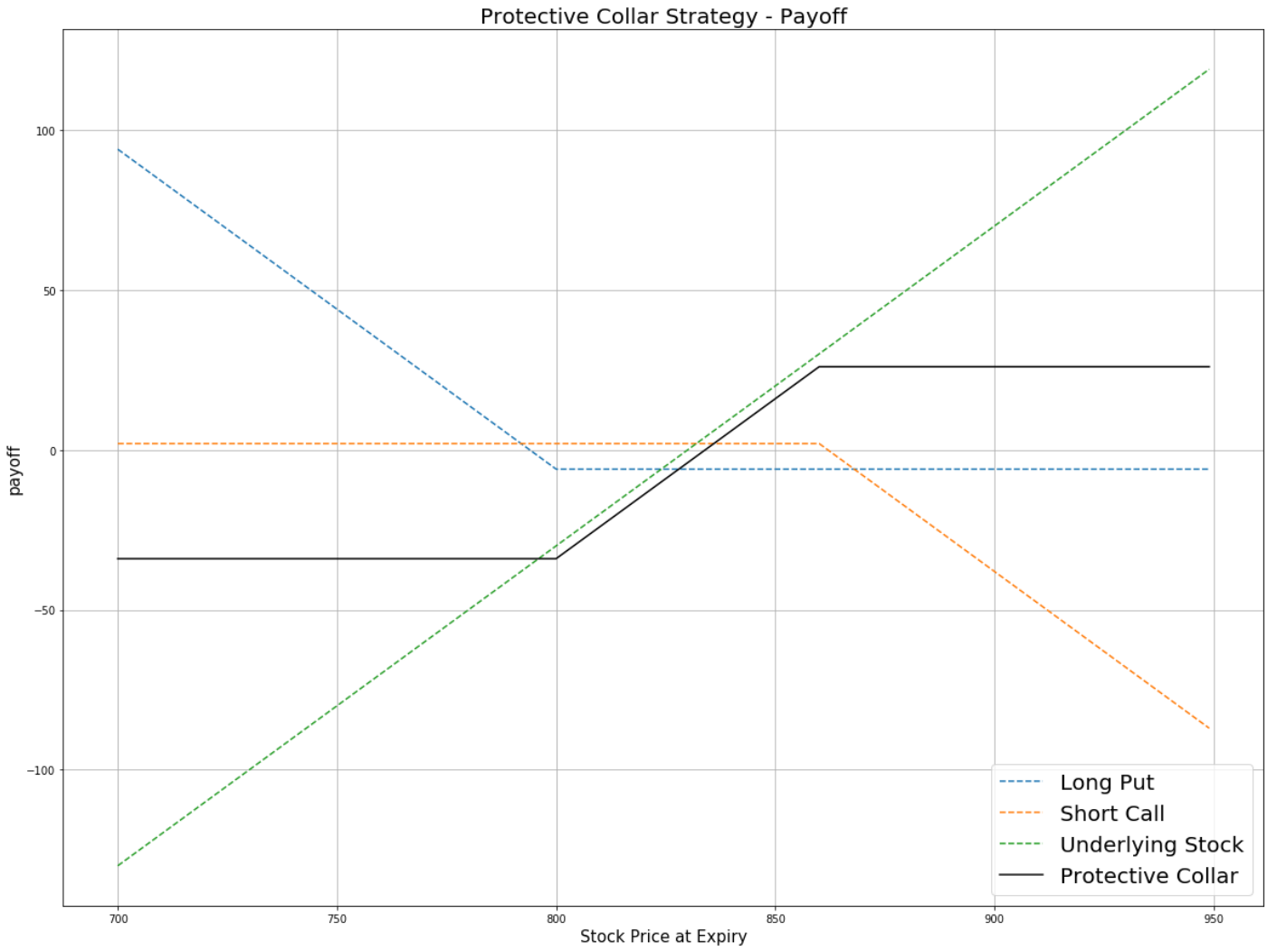
```
PY
self.Sell(call.Symbol, 1)          # Sell the OTM call
self.Buy(put.Symbol, 1)            # Buy the OTM put
self.Buy(self.equity_symbol, 100)  # Buy 100 shares of the underlying stock
```

Strategy Payoff

This is a limited-profit-limited-loss strategy. The payoff is

$$\begin{array}{rcll} C_T & = & (S_T - K^{C})^{+}\\ P_T & = & (K^{P} - S_T)^{+}\\ Payoff_T & = & (S_T - S_0 - C_T + P_T + C_0 - P_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C_T & = & \textrm{Call value at time T}\\ & P_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{Call strike price}\\ & K^{P} & = & \textrm{Put strike price}\\ & Payoff_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & P_0 & = & \textrm{Put price when the trade opened (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{C} - S_T + C_0 - P_0$. It occurs when the underlying price is at or above the strike price of the call at expiration.

The maximum profit is $S_T - K^{P} + C_0 - P_0$. It occurs when the underlying price is at or below the strike price of the put at expiration.

If the Option is American Option, there is risk of early assignment on the sold contract.

Example

The following table shows the price details of the assets in the algorithm:

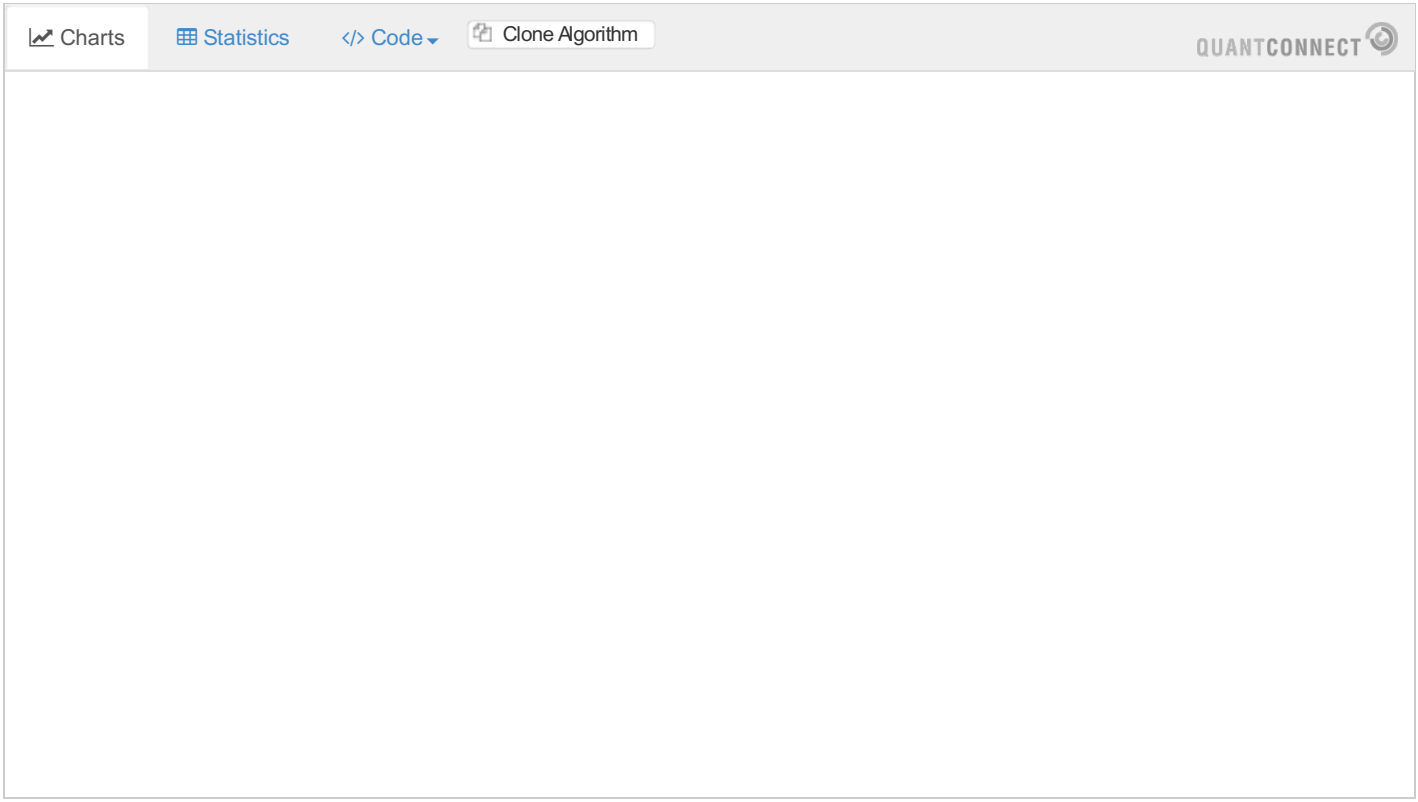
Asset	Price ($)	Strike ($)
Call	2.85	845.00
Put	6.00	822.50
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

$$\begin{array}{rcll} C_T & = & (S_T - K^{C})^{+}\\ & = & (843.19-845.00)^{+}\\ & = & 0\\ P_T & = & (K^{P} - S_T)^{+}\\ & = & (822.50-843.19)^{+}\\ & = & 0\\ Payoff_T & = & (S_T - S_0 - C_T + P_T + C_0 - P_0)\times m - fee\\ & = & (843.19-833.17+0-0+2.85-6.00)\times100-1.00\times3\\ & = & 684\\ \end{array}$$

So, the strategy gains $684.

The following algorithm implements a protective collar Option strategy:



Option Strategies

Straddle

Introduction

Long Straddle is an Options trading strategy that consists of buying an ATM call and an ATM put, where both contracts have the same underlying asset, strike price, and expiration date. This strategy aims to profit from volatile movements in the underlying stock, either positive or negative.

Implementation

Follow these steps to implement the long straddle strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
PY
def Initialize(self) -> None:
    self.SetStartDate(2017, 4, 1)
    self.SetEndDate(2017, 6, 30)
    self.SetCash(100000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(-1, 1, timedelta(30), timedelta(60))
```

2. In the `OnData` method, select the expiration date and strike price of the contracts in the strategy legs.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Select an expiration date
    expiry = sorted(chain, key=lambda contract: contract.Expiry, reverse=True)[0].Expiry

    # Select the ATM strike price
    strikes = [contract.Strike for contract in chain if contract.Expiry == expiry]
    strike = sorted(strikes, key=lambda strike: abs(strike - chain.Underlying.Price))[0]
```

3. In the `OnData` method, call the `OptionStrategies.Straddle` method and then submit the order.

```
PY
option_strategy = OptionStrategies.Straddle(self.symbol, strike, expiry)
self.Buy(option_strategy, 1)
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

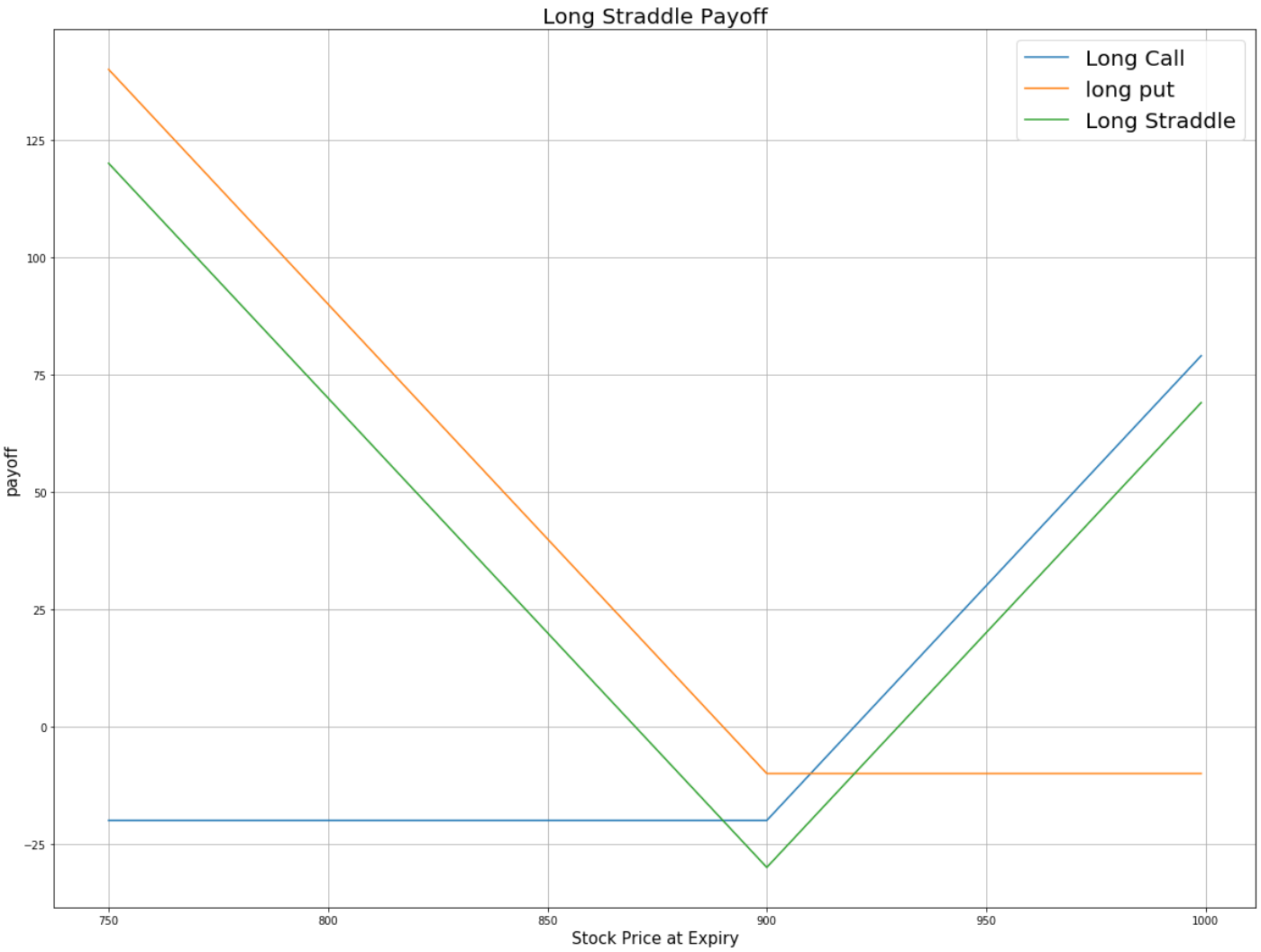
```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{ATM}_T & = & (S_T - K^{C})^{+}\\ P^{ATM}_T & = & (K^{P} - S_T)^{+}\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{ATM}_0 - P^{ATM}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{ATM call strike price}\\ & K^{P} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (debit paid)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited if the underlying price rises to infinity at expiration.

The maximum loss is the net debit paid, $C^{ATM}_0 + P^{ATM}_0$. It occurs when the underlying price is the same at expiration as it was when you opened the trade. In this case, both Options expire worthless.

Example

The following table shows the price details of the assets in the algorithm at Option expiration (2017-05-20):

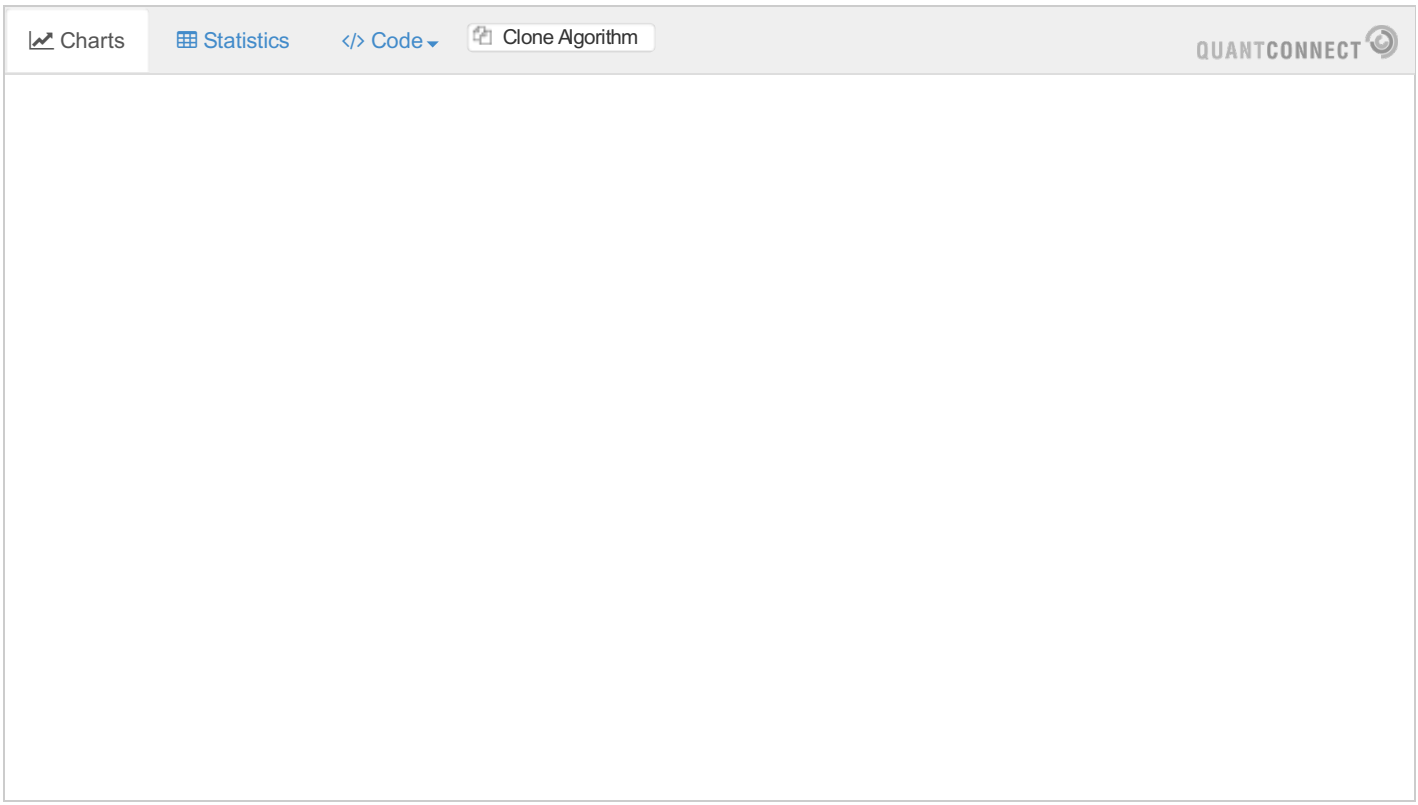
Asset	Price ($)	Strike ($)
Call	22.30	835.00
Put	23.90	835.00
Underlying Equity at expiration	934.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{ATM}_T & = & (S_T - K^{C})^{+}\\ & = & (934.01-835.00)^{+}\\ & = & 98.99\\ P^{ATM}_T & = & (K^{P} - S_T)^{+}\\ & = & (835.00-934.01)^{+}\\ & = & 0\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{ATM}_0 - P^{ATM}_0)\times m - fee\\ & = & (98.99+0-22.3-23.9)\times100-1.00\times2\\ & = & 5277 \end{array}$$

So, the strategy gains $5,277.

The following algorithm implements a long straddle Option strategy:



Option Strategies

Strangle

Introduction

Long Strangle is an Options trading strategy that consists of simultaneously buying an OTM put and an OTM call, where both contracts have the same underlying asset and expiration date. This strategy aims to profit from volatile movements in the underlying stock, either positive or negative.

Compared to a long straddle , the net debit of a long strangle is lower since OTM Options are cheaper. Additionally, the losing range of a long straddle is wider and the strike spread is wider.

Implementation

Follow these steps to implement the long strangle strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```py
def Initialize(self) -> None:
    self.SetStartDate(2017, 4, 1)
    self.SetEndDate(2017, 4, 30)
    self.SetCash(100000)

    option = self.AddOption("GOOG", Resolution.Minute)
    self.symbol = option.Symbol
    option.SetFilter(-5, 5, timedelta(0), timedelta(30))
```

2. In the `OnData` method, select the expiration date and strike prices of the contracts in the strategy legs.

```py
def OnData(self, slice: Slice) -> None:
    if self.Portfolio.Invested: return

    # Get the OptionChain
    chain = slice.OptionChains.get(self.symbol, None)
    if not chain: return

    # Select an expiration date
    expiry = sorted(chain, key=lambda contract: contract.Expiry, reverse=True)[0].Expiry

    # Select the OTM call strike
    strikes = [contract.Strike for contract in chain if contract.Expiry == expiry]
    call_strikes = [contract.Strike for contract in chain
        if contract.Expiry == expiry
        and contract.Right == OptionRight.Call
        and contract.Strike > chain.Underlying.Price]
    if len(call_strikes) == 0: return
    call_strike = min(call_strikes)

    # Select the OTM put strike
    put_strikes = [contract.Strike for contract in chain
        if contract.Expiry == expiry
        and contract.Right == OptionRight.Put
        and contract.Strike < chain.Underlying.Price]
    if len(put_strikes) == 0: return
    put_strike = max(put_strikes)
```

3. In the `OnData` method, call the `OptionStrategies.Strangle` method and then submit the order.

```
option_strategy = OptionStrategies.Strangle(self.symbol, call_strike, put_strike, expiry)
self.Buy(option_strategy, 1)
```

Option strategies synchronously execute by default. To asynchronously execute Option strategies, set the `asynchronous` argument to `False` . You can also provide a tag and order properties to the `Buy` method.

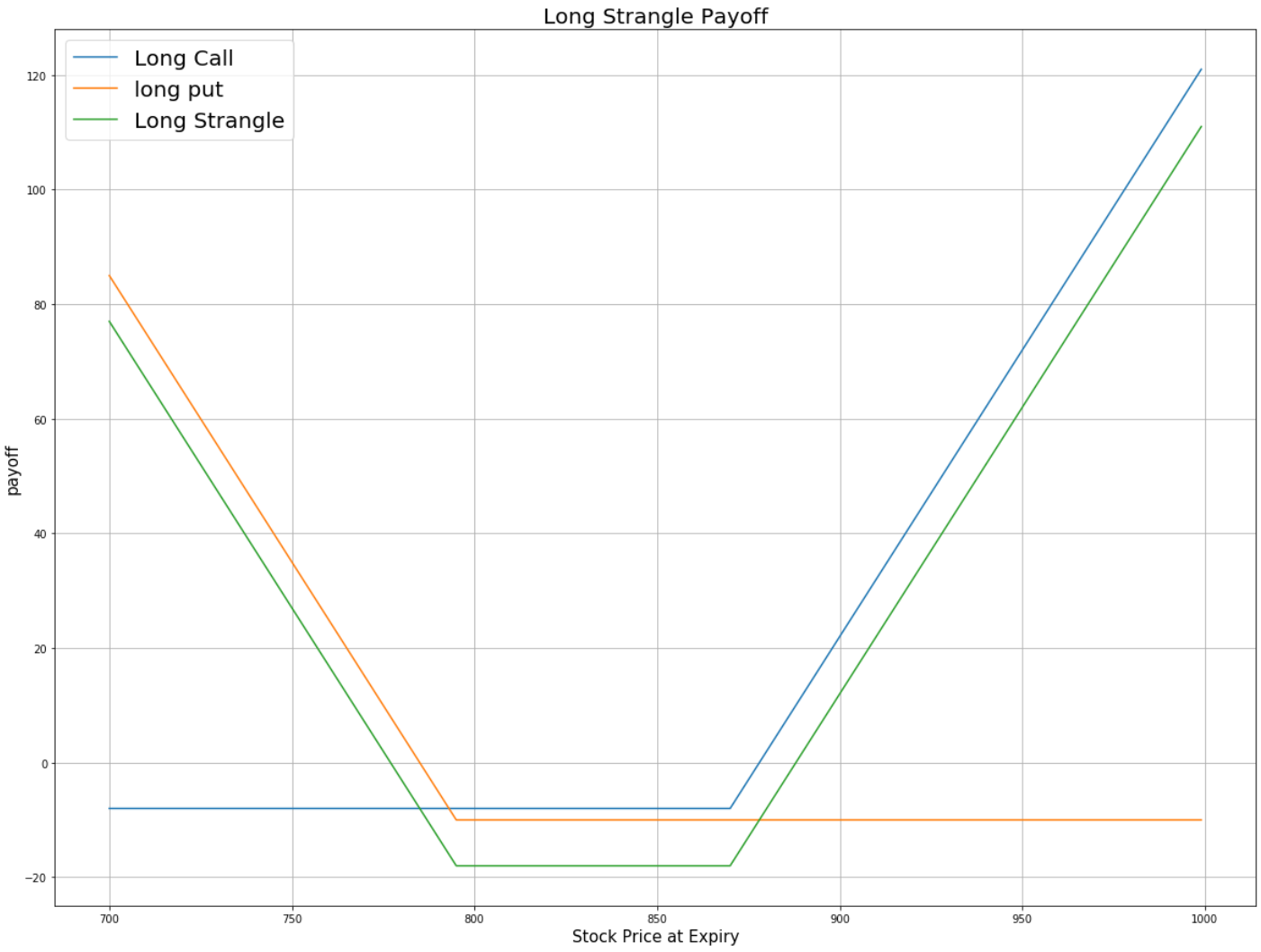
```
self.Buy(option_strategy, quantity, asynchronous, tag, order_properties)
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{C})^{+}\\ P^{OTM}_T & = & (K^{P} - S_T)^{+}\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{OTM}_0 - P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{OTM call strike price}\\ & K^{P} & = & \textrm{OTM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (debit paid)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited if the underlying price rises to infinity at expiration.

The maximum loss is the net debit paid, $C^{OTM}_0 + P^{OTM}_0$. It occurs when the underlying price at expiration is the same as when you opened the trade. In this case, both Options expire worthless.

Example

The following table shows the price details of the assets in the algorithm at Option expiration (2017-04-22):

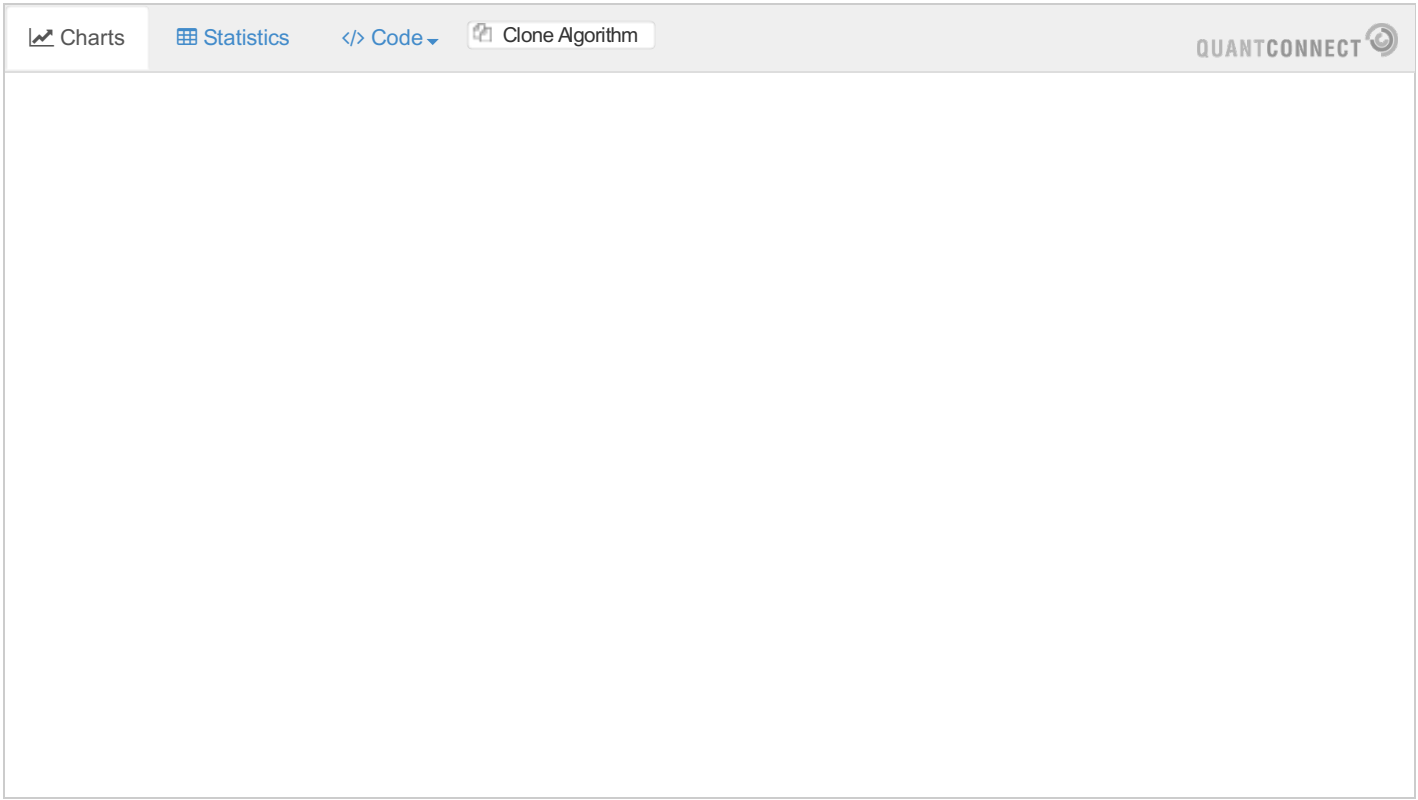
Asset	Price ($)	Strike ($)
Call	8.80	835.00
Put	9.50	832.50
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{C})^{+}\\ & = & (843.19-835.00)^{+}\\ & = & 8.19\\ P^{OTM}_T & = & (K^{P} - S_T)^{+}\\ & = & (832.50-843.19)^{+}\\ & = & 0\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{OTM}_0 - P^{OTM}_0)\times m - fee\\ & = & (8.19+0-8.80-9.50)\times100-1.00\times2\\ & = & -1013 \end{array}$$

So, the strategy losses $1,013.

The following algorithm implements a long strangle Option strategy:



Trading and Orders

Order Properties

Introduction

Order properties enable you to customize how the brokerage executes your orders. The `DefaultOrderProperties` of your algorithm sets the order properties for all of your orders. To adjust the order properties of an order, you can change the `DefaultOrderProperties` or pass an order properties object to the order method.

Time In Force

The `TimeInForce` property determines how long an order should remain open if it doesn't fill. This property doesn't apply to market orders since they usually fill immediately. Time in force is useful to automatically cancel old trades. The following table describes the available TimeInForce options:

Member	Example	Description
`GoodTilCanceled`	`TimeInForce.GoodTilCanceled`	Order is valid until filled (default)
`Day`	`TimeInForce.Day`	Order is valid until filled or the market closes
`GoodTilDate(expiry: datetime)`	`GoodTilDate(datetime(2019, 6, 19, 12, 0, 0))`	Order is valid until filled or the specified expiration time

By default, orders remain open until they are canceled (`TimeInForce.GoodTilCanceled`). To update the value, set the `DefaultOrderProperties.TimeInForce` before you place an order or pass an `orderProperties` argument to the order method.

PY

```
# Set a Limit Order to be good until market close
self.DefaultOrderProperties.TimeInForce = TimeInForce.Day
self.LimitOrder("IBM", 100, 120)

# Set a Limit Order to be good until noon
order_properties = OrderProperties()
order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(2019, 6, 19, 12, 0, 0))
self.LimitOrder("IBM", 100, 120, orderProperties=order_properties)
```

If you trade a market that's open 24 hours a day with daily data, `TimeInForce.Day` won't work because the order cancels at the market close time but your algorithm receives daily data at midnight.

Market on open (MOO) and market on close (MOC) orders don't support the `GoodTilDate` time in force. If you submit a MOO or MOC order with the `GoodTilDate` time in force, LEAN automatically adjusts the time in force to be `GoodTilCanceled` .

The brokerage you use may not support all of the `TimeInForce` options. To see the options that your brokerage

supports, see the **Orders** section of the brokerage model documentation .

Brokerage-Specific Properties

Some brokerages support additional order properties so you can customize how your orders execute. Some examples include the following order properties:

- Financial Advisor group orders
- An `OutsideRegularTradingHours` property to let orders fill during pre-market and post-market trading hours
- A `PostOnly` property to force an order to only add liquidity to a market
- A `Hidden` property to make an order not show on the order book
- A `ReduceOnly` property to signal the order must only decrease your position size
- `FeeInBase` and `FeeInQuote` properties to set which currency you pay fees in for a Crypto trade

To view the order properties your brokerage supports, see the **Orders** section of the brokerage model documentation .

Tags

You can tag orders to aid your strategy development. Tags can be any string of up to 100 characters. Tags aren't part of the `OrderProperties` object, but they are a property of the `Order` class you can set. To set an order tag, pass it as an argument when you create the order or use the order update methods.

```py
 # Tag an order on creation
ticket = self.LimitOrder("SPY", 100, 221.05, "Original tag")

# Update the tag with UpdateTag
ticket.UpdateTag("Updated tag")

# Update the tag with UpdateOrderFields
update_settings = UpdateOrderFields()
update_settings.Tag = "Another tag"
ticket.Update(update_settings)
```

Trading and Orders

Order Events

Introduction

An `OrderEvent` object represents an update to the state of an order. As the state of your orders change, we notify your algorithm with `OrderEvent` objects through the `OnOrderEvent` and `OnAssignmentOrderEvent` event handlers.

Track Order Events

Each order generates events over its life as its status changes. Your algorithm receives these events through the `OnOrderEvent` and `OnAssignmentOrderEvent` methods. The `OnOrderEvent` event handler receives all order events. The `OnAssignmentOrderEvent` receives order events for Option assignments. The event handlers receive an `OrderEvent` object, which contains information about the order status.

```
PY
def OnOrderEvent(self, orderEvent: OrderEvent) -> None:
    order = self.Transactions.GetOrderById(orderEvent.OrderId)
    if orderEvent.Status == OrderStatus.Filled:
        self.Debug(f"{self.Time}: {order.Type}: {orderEvent}")

def OnAssignmentOrderEvent(self, assignmentEvent: OrderEvent) -> None:
    self.Log(str(assignmentEvent))
```

To get a list of all `OrderEvent` objects for an order, call the `OrderEvents` method of the order ticket .

```
PY
order_events = order_ticket.OrderEvents()
```

If you don't have the order ticket, get the order ticket from the TransactionManager .

Order States

Orders can have any of the following states:

Event Attributes

The `OnOrderEvent` and `OnAssignmentOrderEvent` event handlers in your algorithm receive `OrderEvent` objects, which have the following attributes:

Trading and Orders

Order Errors

Introduction

If an error occurs with one of your orders, LEAN logs the error message and stops executing.

Types of Order Errors

Errors can occur when you place orders or after you place them.

Errors at Order Submission

The following errors can occur when you submit an order:

- `AlgorithmWarmingUp`
- `BrokerageFailedToSubmitOrder`
- `BrokerageModelRefusedToSubmitOrder`
- `ConversionRateZero`
- `ExceededMaximumOrders`
- `ExceedsShortableQuantity`
- `ExchangeNotOpen`
- `ForexBaseAndQuoteCurrenciesRequired`
- `ForexConversionRateZero`
- `InsufficientBuyingPower`
- `MarketOnCloseOrderTooLate`
- `MissingSecurity`
- `NonExercisableSecurity`
- `NonTradableSecurity`
- `OrderAlreadyExisits`
- `OrderQuantityLessThanLotSize`
- `OrderQuantityZero`
- `PreOrderChecksError`
- `QuoteCurrencyRequired`
- `SecurityHasNoData`
- `UnsupportedRequestType`
- `EuropeanOptionNotExpiredOnExercise`

Errors After Order Submission

The following errors can occur for active orders:

- `BrokerageFailedToUpdateOrder`
- `BrokerageModelRefusedToUpdateOrder`

- `InvalidNewOrderStatus`
- `InvalidOrderStatus`
- `InvalidRequest`
- `RequestCanceled`
- `UnableToFindOrder`

Common Errors

There are a few common order errors you may experience.

Why is my order converted to a market on open order?

If you place a market order when the market is closed, LEAN automatically converts the order into market on open order. This most commonly happens when you use daily or hourly data, which your algorithm can receive when the market is closed. Your algorithm receives daily bars at midnight and receives the last hourly bar of each day at 4 PM Eastern Time (ET). To avoid LEAN from converting your market order to market on open orders, submit your market orders when the market is open. To check if the market is open, call the `IsMarketOpen` method.

```
PY
if self.IsMarketOpen(self.symbol):
    self.MarketOrder(self.symbol, quantity)
```

Why am I seeing the "stale price" warning?

Stale fills occur when you fill an order with price data that is timestamped an hour or more into the past. Stale fills usually only occur if you trade illiquid assets or if your algorithm uses daily data but you trade intraday with Scheduled Events. If your order is filled with stale data, the fill price may not be realistic. The pre-built fill models can only fill market orders with stale data. To adjust the length of time that needs to pass before an order is considered stale, set the `StalePriceTimeSpan` setting.

```
PY
self.Settings.StalePriceTimeSpan = timedelta(minutes=10)
```

Why do I get "Backtest Handled Error: The security with symbol '/ES' is marked as non-tradable"?

This error occurs when you place an order for a continuous Futures contract, which isn't a tradable security. To fix the issue, place the order for a specific Futures contract. To access the currently selected contract in the continuous contract series, use the `Mapped` property of the `Future` object.

Error Code Reference

The `OrderError` enumeration has the following members:

Order Response Error Reference

The following sections explain why each `OrderResponseErrorCode` occurs and how to avoid it.

None

The `OrderResponseErrorCode.None` (0) error means there is no order response error.

Processing Error

The `OrderResponseErrorCode.ProcessingError` (-1) error occurs in the following situations:

- When you submit a new order, but LEAN throws an exception while checking if you have sufficient buying power for the order.
- When you try to update or cancel an order, but LEAN throws an exception.

To investigate this order response error further, see the `HandleSubmitOrderRequest` , `UpdateOrder` , and `CancelOrder` methods of the BrokerageTransactionHandler in the LEAN GitHub repository.

Order Already Exists

The `OrderResponseErrorCode.OrderAlreadyExists` (-2) error occurs when you submit a new order but you already have an open order or a completed order with the same order ID. This order response error usually comes from a concurrency issue.

To avoid this order response, don't place two asynchronous orders at the same time. GitHub Issue #6736 may address the underlying problem.

Insufficient Buying Power

The `OrderResponseErrorCode.InsufficientBuyingPower` (-3) error occurs when you place an order but the buying power model determines you can't afford it.

This error commonly occurs when you place a market on open order with daily data. If you place the order with `SetHoldings` or use `CalculateOrderQuantity` to determine the order quantity, LEAN calculates the order quantity based on the market close price. If the open price on the following day makes your order more expensive, then you may have insufficient buying power. To avoid the order response error in this case, either use intraday data and place trades when the market is open or adjust your buying power buffer .

```
PY
self.Settings.FreePortfolioValuePercentage = 0.05
```

Brokerage Model Refused to Submit Order

The `OrderResponseErrorCode.BrokerageModelRefusedToSubmitOrder` (-4) error occurs when you place an order but the brokerage model determines it's invalid. The brokerage model usually checks your order meets the following requirements before sending it to the brokerage:

- Supported security types
- Supported order types and their respective requirements
- Supported time in force options
- The order size is larger than the minimum order size

Each brokerage model can have additional order requirements that the brokerage declares. To avoid this order response error, see the **Orders** section of the brokerage model documentation .

To investigate this order response error further, see the `CanSubmitOrder` method definition of your brokerage model .

This order response error occurs when the `CanSubmitOrder` method returns `False` .

Brokerage Failed to Submit Order

The `OrderResponseErrorCode.BrokerageFailedToSubmitOrder` (-5) error occurs when you place an order but the brokerage implementation fails to submit the order to your brokerage.

To investigate this order response error further, see the `PlaceOrder` method definition of your brokerage or the BacktestingBrokerage in the LEAN GitHub repository. This order response error occurs when the `PlaceOrder` method throws an error or returns `false` .

Brokerage Failed to Update Order

The `OrderResponseErrorCode.BrokerageFailedToUpdateOrder` (-6) error occurs when you try to update an order but the brokerage implementation fails to submit the order update request to your brokerage.

To avoid this order response error, see the **Orders** section of the brokerage model documentation .

To investigate this order response error further, see the `UpdateOrder` method definition of your brokerage or the BacktestingBrokerage in the LEAN GitHub repository. This order response error occurs when the `UpdateOrder` method throws an error or returns `false` .

Brokerage Failed to Cancel Order

The `OrderResponseErrorCode.BrokerageFailedToCancelOrder` (-8) error occurs when you try to cancel an order but the brokerage implementation fails to submit the cancel request to your brokerage.

To investigate this order response error further, see the `CancelOrder` method definition of your brokerage or the BacktestingBrokerage in the LEAN GitHub repository. This order response error occurs when `CancelOrder` method throws an error or returns `false` .

Invalid Order Status

The `OrderResponseErrorCode.InvalidOrderStatus` (-9) error occurs when you try to update or cancel an order but the order is already complete. An order is complete if it has `OrderStatus.Filled` , `OrderStatus.Canceled` , or `OrderStatus.Invalid` .

To avoid this order response error, check `Status` of an order ticket or order event before you update or cancel the order.

```
PY
if not OrderExtensions.IsClosed(order_ticket.Status):
    order_ticket.Cancel()
```

Unable to Find Order

The `OrderResponseErrorCode.UnableToFindOrder` (-10) error occurs when you try to place, update, or cancel an order, but the `BrokerageTransactionHandler` doesn't have a record of the order ID.

To investigate this order response error further, see BrokerageTransactionHandler.cs in the LEAN GitHub repository. This order response error occurs when the `BrokerageTransactionHandler` can't find the order ID in it's

`_completeOrders` or `_completeOrderTickets` dictionaries.

Order Quantity Zero

The `OrderResponseErrorCode.OrderQuantityZero` (-11) error occurs when you place an order that has zero quantity or when you update an order to have a zero quantity. This error commonly occurs if you use the SetHoldings method but the portfolio weight you provide to the method is too small to translate into a non-zero order quantity.

To avoid this order response error, check if the quantity of the order is non-zero before you place the order. If you use the `SetHoldings` method, replace it with a combination of the CalculateOrderQuantity and MarketOrder methods.

```
PY
quantity = self.CalculateOrderQuantity(self.symbol, 0.05)
if quantity:
    self.MarketOrder(self.symbol, quantity)
```

Unsupported Request Type

The `OrderResponseErrorCode.UnsupportedRequestType` (-12) error occurs in the following situations:

- When you try to exercise an Option contract for which you hold a short position
- When you try to exercise more Option contracts than you hold

To avoid this order response error, check the quantity of your holdings before you try to exercise an Option contract.

```
PY
holding_quantity = self.Portfolio[self.contract_symbol].Quantity
if holding_quantity > 0:
    self.ExerciseOption(self.contract_symbol, max(holding_quantity, exercise_quantity))
```

Missing Security

The `OrderResponseErrorCode.MissingSecurity` (-14) error occurs when you place an order for a security but you don't have a subscription for the security in your algorithm.

To avoid this order response error, create a subscription for each security you want to trade. To create subscriptions, see the Requesting Data page of the documentation for each asset class .

Exchange Not Open

The `OrderResponseErrorCode.ExchangeNotOpen` (-15) error occurs in the following situations:

- When you try to exercise an Option while the exchange is not open

  To avoid the order response error in this case, check if the exchange is open before you exercise an Option contract.

  ```
  PY
  if self.IsMarketOpen(self.contract_symbol):
      self.ExerciseOption(self.contract_symbol, quantity)
  ```

- When you try to place a market on open order for a Futures contract or a Future Option contract

Security Price Zero

The `OrderResponseErrorCode.SecurityPriceZero` (-16) error occurs when you place an order or exercise an Option contract while the security price is 0. The security price can be 0 for the following reasons:

- The data is missing

  Investigate if it's a data issue. If it is a data issue, report it .

- The algorithm hasn't received data for the security yet

  If you subscribe to a security and place an order for the security in the same time step , you'll get this error. To avoid this order response error, initialize the security price with the last known price.

  ```
  self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel,
  FuncSecuritySeeder(self.GetLastKnownPrices)))
  ```
  PY

Forex Base and Quote Currencies Required

The `OrderResponseErrorCode.ForexBaseAndQuoteCurrenciesRequired` (-17) error occurs when you place a trade for a Forex or Crypto pair but you don't have the base currency and quote currency in your cash book . This error should never occur. If it does, create a bug report.

Forex Conversion Rate Zero

The `OrderResponseErrorCode.ForexConversionRateZero` (-18) error occurs when you place a trade for a Forex or Crypto pair and LEAN can't convert the value of the base currency to your account currency. This error usually indicates a lack of data. Investigate the data and if there is some missing, report it .

Security Has No Data

The `OrderResponseErrorCode.SecurityHasNoData` (-19) error occurs when you place an order for a security before your algorithm receives any data for it. If you subscribe to a security and place an order for the security in the same time step , you'll get this error. To avoid this order response error, initialize the security price with the last known price.

```
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))
```
PY

Exceeded Maximum Orders

The `OrderResponseErrorCode.ExceededMaximumOrders` (-20) error occurs when exceed your order quota in a backtest. The number of orders you can place in a single backtest depends on the tier of your organization . The following table shows the number of orders you can place on each tier:

Tier	Orders Quota
Free	10K
Quant Researcher	10M
Team	Unlimited
Trading Firm	Unlimited
Institution	Unlimited

To avoid this order response error, reduce the number of orders in your backtest or upgrade your organization .

Market on Close Order Too Late

The `OrderResponseErrorCode.MarketOnCloseOrderTooLate` (-21) error occurs when you try to place a market on close (MOC) order too early in the trading day.

To avoid this order response error, place the MOC order closer to the market close or adjust the submission time buffer. By default, you must place MOC orders at least 15.5 minutes before the close, but some exchanges let you submit them closer to the market closing time. To adjust the buffer period that's required, set the `MarketOnCloseOrder.SubmissionTimeBuffer` property.

```
MarketOnCloseOrder.SubmissionTimeBuffer = timedelta(minutes=10)
```

Invalid Request

The `OrderResponseErrorCode.InvalidRequest` (-22) error occurs when you try to cancel an order multiple times.

To avoid this order response error, only try to cancel an order one time.

Request Canceled

The `OrderResponseErrorCode.RequestCanceled` (-23) error occurs when you try to cancel an order multiple times.

To avoid this order response error, only try to cancel an order one time.

Algorithm Warming Up

The `OrderResponseErrorCode.AlgorithmWarmingUp` (-24) error occurs in the following situations:

- When you try to place, update, or cancel an order during the warm-up period
- When the Option assignment simulator assigns you to an Option during the warm-up period

To avoid this order response error, only manage orders after the warm-up period ends. To avoid trading during the warm-up period, add an `IsWarmingUp` guard to the top of the `OnData` method.

```
if self.IsWarmingUp: return
```

Brokerage Model Refused to Update Order

The `OrderResponseErrorCode.BrokerageModelRefusedToUpdateOrder` (-25) error occurs in backtests when you try to update an order in a way that the brokerage model doesn't support.

To avoid this issue, see the **Orders** section of the brokerage model documentation to check its order requirements.

To investigate this order response error further, see the `CanUpdateOrder` method definition of your brokerage model .

Quote Currency Required

The `OrderResponseErrorCode.QuoteCurrencyRequired` (-26) error occurs when you place an order for a Forex or Crypto pair and don't have the quote currency of the pair in your cash book . This error should never occur. If it does, create a bug report.

Conversion Rate Zero

The `OrderResponseErrorCode.ConversionRateZero` (-27) error occurs when you place an order for a Forex or Crypto pair and LEAN can't convert the value of the quote currency in the pair to your account currency. This order response error usually indicates a lack of data. Investigate the data and if there is data missing, report it .

Non-Tradable Security

The `OrderResponseErrorCode.NonTradableSecurity` (-28) error occurs when you place an order for a security that's not tradable . To avoid this order response error, check if a security is tradable before you trade it.

```py
if self.Securities[self.symbol].IsTradable:
    self.MarketOrder(self.symbol, quantity)
```

Non-Exercisable Security

The `OrderResponseErrorCode.NonExercisableSecurity` (-29) error occurs when you call the ExerciseOption method with a `Symbol` that doesn't reference an Option contract.

Order Quantity Less Than Lot Size

The `OrderResponseErrorCode.OrderQuantityLessThanLotSize` (-30) error occurs when you place an order with a quantity that's less than the lot size of the security.

To avoid this order response error, check if the order quantity is greater than or equal to the security lot size before you place an order.

```py
lot_size = self.Securities[self.symbol].SymbolProperties.LotSize
if quantity >= lot_size:
    self.MarketOrder(self.symbol, quantity)
```

Exceeds Shortable Quantity

The `OrderResponseErrorCode.ExceedsShortableQuantity` (-31) error occurs when you place an order to short a security but the shortable provider of the brokerage model states there isn't enough shares to borrow. For a full

example of this error, clone and run this backtest .

To avoid this order response error, check if there are enough shares available before you place an order to short a security.

```
available_to_borrow = self.BrokerageModel.GetShortableProvider().ShortableQuantity(self.symbol, self.Time)
if available_to_borrow == None or quantity_to_borrow <= available_to_borrow:
    self.MarketOrder(self.symbol, -quantity_to_borrow)
```

PY

Invalid New Order Status

The `OrderResponseErrorCode.InvalidNewOrderStatus` (-32) error occurs in live trading when you try to update or cancel an order while it still has `OrderStatus.New` status.

To avoid this order response error, check the `Status` property of the order ticket or order event before you update or cancel an order.

```
if self.order_ticket.Status != OrderStatus.New:
    self.order_ticket.Cancel()
```

PY

European Option Not Expired on Exercise

The `OrderResponseErrorCode.EuropeanOptionNotExpiredOnExercise` (-33) error occurs when you try to exercise a European Option contract before its expiry date.

To avoid this order response error, check the type and expiry date of the contract before you exercise it.

```
if (self.contract_symbol.ID.OptionStyle == OptionStyle.European && self.contract_symbol.ID.Date ==
self.Time.Date)
{
    self.ExerciseOption(self.contract_symbol, quantity);
}
```

PY

Trading and Orders

Trade Statistics

Introduction

The `TradeBuilder` tracks the trades of your algorithm and calculates some statistics. You can adjust how it defines a trade and how it matches offsetting order fills.

Set Trade Builder

To set the `TradeBuilder` , in the `Initialize` method, call the `SetTradeBuilder` method.

```
self.SetTradeBuilder(TradeBuilder(groupingMethod, matchingMethod))
```

The following table describes the arguments the `TradeBuilder` constructor accepts:

Argument	Data Type	Description	Default Value
`groupingMethod`	`FillGroupingMethod`	The method used to group order fills into trades	
`matchingMethod`	`FillMatchingMethod`	The method used to match offsetting order fills	

The `FillGroupingMethod` enumeration has the following members:

The `FillMatchingMethod` enumeration has the following members:

Default Behavior

The default `TradeBuilder` uses `FillGroupingMethod.FillToFill` and `FillMatchingMethod.FIFO` .

Check Open Positions

To check if you have a position open for a security as defined by the `FillGroupingMethod` , call the `HasOpenPosition` method.

```
has_open_position = self.TradeBuilder.HasOpenPosition(self.symbol)
```

Get Closed Trades

To get your closed trades, use the `ClosedTrades` property.

```py
trades = self.TradeBuilder.ClosedTrades
```

This property returns a list of `Trade` objects, which have the following attributes:

Trading and Orders

Trading Calendar

Introduction

The `TradingCalendar` contains a variety of events relevant to the assets in your algorithm. We source the event data from Quantlib .

Trading Day Properties

`TradingDay` objects represent the trading events of a single day. They have the following properties:

Get a Single Trading Day

To get the trading events of the current day, call the `GetTradingDay` method with no arguments.

```
PY
trading_day = self.TradingCalendar.GetTradingDay()
```

To get the trading events of a specific day, pass a `datetime` object to the `GetTradingDay` method.

```
PY
trading_day = self.TradingCalendar.GetTradingDay(datetime(2022, 6, 1))
```

Get Trading Days

To get all the trading events across a range of dates, pass start and end `datetime` objects to the `GetTradingDays` method.

```
PY
start_date = datetime(2022, 6, 1)
end_date = datetime(2022, 7, 1)
trading_days = self.TradingCalendar.GetTradingDays(start_date, end_date)
```

To get specific trading events across a range of dates, pass a `TradingDayType` enumeration member and the dates to the `GetDaysByType` method.

```
PY
trading_days = self.TradingCalendar.GetDaysByType(TradingDayType.OptionExpiration, start_date, end_date)
```

The `TradingDayType` enumeration has the following members:

Trading and Orders

Financial Advisors

Introduction

Financial Advisor accounts enable certified professionals to use a single trading algorithm to manage several client accounts. Our Interactive Brokers integration enables you to place FA group orders.

Group Routing

To place trades using a subset of client accounts, create Account Groups in Trader Workstation and then define the `InteractiveBrokersOrderProperties` when you create orders.

```py
self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
self.DefaultOrderProperties.FaGroup = "TestGroupEQ"
self.DefaultOrderProperties.FaMethod = "EqualQuantity"
self.DefaultOrderProperties.FaProfile = "TestProfileP"
self.DefaultOrderProperties.Account = "DU123456"
```

SecurityHolding objects aggregate your positions across all the account groups. If you have two groups where group A has 10 shares of SPY and group B has -10 shares of SPY, then `self.Portfolio["SPY"].Quantity` is zero.

Allocation Methods

The following table shows the supported allocation methods for FA group orders:

`FaMethod`	**Description**
`"EqualQuantity"`	Distributes shares equally between all accounts in the group. If you use this method, specify an order quantity.
`"NetLiq"`	Distributes shares based on the net liquidation value of each account. The system calculates ratios based on the net liquidation value in each account and allocates shares based on these ratios. If you use this method, specify an order quantity.
`"AvailableEquity"`	Distributes shares based on the amount of available equity in each account. The system calculates ratios based on the available equity in each account and allocates shares based on these ratios. If you use this method, specify an order quantity.
`"PctChange"`	Increases or decreases an already existing position. Positive percents increase positions and negative percents decrease positions. If you use this method, specify a percent instead of an order quantity.

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
    self.DefaultOrderProperties.FaGroup = "TestGroupEQ"
    self.DefaultOrderProperties.FaMethod = "EqualQuantity"
    self.DefaultOrderProperties.FaProfile = "TestProfileP"
    self.DefaultOrderProperties.Account = "DU123456"

def OnData(self, slice: Slice) -> None:
    # Override the default order properties
    # "NetLiq" requires a order size input
    order_properties = InteractiveBrokersOrderProperties()
    order_properties.FaMethod = "NetLiq"
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    # "AvailableEquity" requires a order size input
    order_properties.FaMethod = "AvailableEquity"
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    # "PctChange" requires a percentage of portfolio input
    order_properties.FaMethod = "PctChange"
    self.SetHoldings(self.symbol, pct_portfolio, orderProperties=order_properties)
```

Subscription Requirements

To use FA group orders through our Interactive Brokers integration, you need to connect as a member of a Trading Firm and Institution organization. If you aren't currently on either of these tiers, upgrade your organization .

Reality Modeling

Reality Modeling

Key Concepts

Introduction

Reality models make backtests as realistic as possible to how the strategy would perform in live trading. These reality models model the behavior of things like the portfolio, brokerage, fills, slippage, options, and strategy capacity . Some reality models are set on a security basis and some are set at the portfolio level. The default models assume you trade highly liquid assets. If you trade high volumes or on illiquid assets, you should create custom reality models to be more realistic.

Security Level Models

LEAN's philosophy is to make the models per security as much as possible. The following models are security-level models:

- Fill
- Slippage
- Fee
- Brokerages
- Buying Power
- Settlement
- Option models
  - Pricing
  - Volatility
  - Exercise
  - Assignment
- Margin Interest Rate

To set a security-level reality model, call the set reality model method on the `Security` object. To get the correct method, see the preceding documentation page for each type of model.

```
# Set IBM to have a constant $1 transaction fee
self.Securities["IBM"].SetFeeModel(ConstantFeeModel(1))
```

PY

You can also set the security-specific models inside a security initializer .

```py
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetFeeModel(ConstantFeeModel(1))
```

Portfolio Level Models

A portfolio is a semi-closed system where the state updates over time as the value and quantity of assets in the portfolio fluctuate. It's a semi-closed system because the portfolio usually incurs transaction fees and you can add and deposit capital from the portfolio. LEAN models the portfolio holdings to track and update the average price of each asset with every transaction.

At the portfolio level, you can set the margin call model .

```py
self.Portfolio.SetMarginCallModel(DefaultMarginCallModel(self.Portfolio, self.DefaultOrderProperties))
```

Reality Modeling

Trade Fills

Trade Fills

Key Concepts

Introduction

A trade fill is the quantity and price at which your brokerage executes your order in the market. Fill models model how each type of order fills to accurately simulate the behavior of a real brokerage. Fill models determine the price and quantity of your fills, can incorporate spread costs, and work with the slippage model to add slippage into the fill price. If you trade US Equities, our built-in fill models can fill your orders at the official opening and closing auction prices .

Set Models

The brokerage model of your algorithm automatically sets the fill model for each security, but you can override it. To manually set the fill model of a security, call the `SetFillModel` method on the Security object.

PY

```
# In Initialize
security = self.AddEquity("SPY")
security.SetFillModel(ImmediateFillModel())
```

You can also set the fill model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

PY

```
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetFillModel(ImmediateFillModel())
```

To view all the pre-built fill models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the fill model of each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the EquityFillModel for Equities, the FutureFillModel for Futures, the FutureOptionFillModel for Future Options, and the ImmediateFillModel for all other asset classes.

Model Structure

Fill Models should extend the `FillModel` class. To implement your own fill model, override the methods in the `FillModel` class you wish to change. The class has a dedicated method for each order type. The method receives a `Security` and `Order` object and returns an `OrderEvent` object that contains information about the order status, fill quantity, and errors.

```
PY
class MyFillModel(FillModel):

    def MarketFill(self, asset: Security, order: MarketOrder) -> OrderEvent:
        return super().MarketFill(asset, order)

    def LimitFill(self, asse: Securityt, order: LimitOrder) -> OrderEvent:
        return super().LimitFill(asset, order)

    def LimitIfTouchedFill(self, asset: Security, order: LimitIfTouchOrder) -> OrderEvent:
        return super().LimitIfTouchedFill(asset, order)

    def StopMarketFill(self, asset: Security, order: StopMarketOrder) -> OrderEvent:
        return super().StopMarketFill(asset, order)

    def StopLimitFill(self, asset: Security, order: StopLimitOrder) -> OrderEvent:
        return super().StopLimitFill(asset, order)

    def MarketOnOpenFill(self, asset: Security, order: MarketOnOpenOrder) -> OrderEvent:
        return super().MarketOnOpenFill(asset, order)

    def MarketOnCloseFill(self, asset: Security, order: MarketOnCloseOrder) -> OrderEvent:
        return super().MarketOnCloseFill(asset, order)
```

Partial Fills

In live trading, your orders can partially fill. For example, if you have a buy limit order at the bid price for 100 shares and someone sells 10 shares with a market order, your order is partially filled. In backtests, the pre-built fill models assume orders completely fill. To simulate partial fills in backtests, create a custom fill model.

Stale Fills

Stale fills occur when you fill an order with price data that is timestamped an hour or more into the past. Stale fills usually only occur if you trade illiquid assets or if your algorithm uses daily data but you trade intraday with Scheduled Events. If your order is filled with stale data, the fill price may not be realistic. The pre-built fill models can only fill market orders with stale data. To adjust the length of time that needs to pass before an order is considered stale, set the `StalePriceTimeSpan` setting.

```
PY
self.Settings.StalePriceTimeSpan = timedelta(minutes=10)
```

Examples

Demonstration Algorithms
CustomModelsAlgorithm.py Python CustomPartialFillModelAlgorithm.py Python ForwardDataOnlyFillModelAlgorithm.py Python

Trade Fills

Supported Models

Fill models determine the price and quantity of your fills. The following fill models are available:

Equity Model

Equity

Future Model

Futures

Future Option Model

Future Options

Immediate Model

All asset classes

Latest Price Model

All asset classes

See Also

Reality Modeling
Order Types

Supported Models

Equity Model

Introduction

The `EquityFillModel` is the default fill model if you trade Equity assets with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
security.SetFillModel(EquityFillModel())
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following sections explain the fill logic of each order given these factors.

To view the implementation of this model, see the LEAN GitHub repository .

Market Orders

The model fills buy market orders at the best effort ask price plus slippage and fills sell market orders at the best effort bid price minus slippage.

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

The model only fills market orders during regular trading hours.

Limit Orders

To fill limit orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent back of trade

ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

Once the model has a valid best effort `TradeBar` , it can fill the order. The following table shows the fill condition and fill price of limit orders. The model only fills the order once the fill condition is met.

Order Direction	Fill Condition	Fill Price
Buy	low price < limit price	min(open price, limit price)
Sell	high price > limit price	max(open price, limit price)

The model only fills limit orders when the exchange is open.

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

To fill limit if touched orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent back of trade ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

After the model has a valid best effort `TradeBar` , it can check if the trigger price has been touched. The following table describes the trigger condition of limit if touched orders for each order direction:

Order Direction	Trigger Condition
Buy	low price <= trigger price
Sell	high price >= trigger price

Once the limit if touched order triggers, the model starts to check if it should fill the order. The following table shows the fill condition and fill price of limit if touched orders. The model only fills the order once the fill condition is met

Order Direction	Fill Condition	Fill Price
Buy	Best effort ask price <= limit price	min(best effort ask price, limit price)
Sell	Best effort bid price >= limit price	max(best effort bid price, limit price)

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

The model only fills limit orders when the exchange is open.

The model won't trigger or fill limit if touched orders with stale data .

Stop Market Orders

To fill stop market orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent back of trade ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

Once the stop condition is met, the model fills the orders and sets the fill price.

Once the model has a valid best effort `TradeBar` , it can fill the order. The following table shows the stop condition and fill price of stop market orders. The model only fills the order once the stop condition is met.

Order Direction	Fill Condition	Fill Price
Buy	high price >= stop price	max(open price, stop price) + slippage
Sell	low price <= stop price	min(open price, stop price) - slippage

The model only fills stop market orders during regular trading hours.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The fill logic of stop limit orders depends on the data format of the security subscription and the order direction. The following table shows the fill price of stop limit orders given these factors. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

The following table describes how the fill model processes the order given the data format and order direction. Once

the stop condition is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

The model only fills stop limit orders when the exchange is open.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order direction:

Data Format	Order Direction	Fill Price
`Tick`	Buy	If the model receives the official opening auction price within one minute, the order fills at official open price + slippage. After one minute, the order fills at the most recent trade price + slippage. If the security doesn't trade within the first two minutes, the order fills at the best effort ask price + slippage.
`Tick`	Sell	If the model receives the official opening auction price within one minute, the order fills at the official open price - slippage. After one minute, the order fills at the most recent trade price - slippage. If the security doesn't trade within the first two minutes, the order fills at the best effort bid price - slippage.
`TradeBar`	Buy	Open price + slippage
`TradeBar`	Sell	Open price - slippage
`QuoteBar`	Buy	Best effort ask price + slippage
`QuoteBar`	Sell	Best effort bid price - slippage

The model checks the data format in the following order:

1. `Tick`
2. `TradeBar`
3. `QuoteBar`

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order direction:

Data Format	Order Direction	Fill Price
`Tick`	Buy	If the model receives the official closing auction price within one minute after the close, the order fills at official close price + slippage. After one minute, the order fills at the most recent trade price + slippage. If the security doesn't trade within the first two minutes, the order fills at the best effort ask price + slippage.
`Tick`	Sell	If the model receives the official closing auction price within one minute after the close, the order fills at the official close price - slippage. After one minute, the order fills at the most recent trade price - slippage. If the security doesn't trade within the first two minutes after the close, the order fills at the best effort bid price - slippage.
`TradeBar`	Buy	Open price + slippage
`TradeBar`	Sell	Open price - slippage
`QuoteBar`	Buy	Best effort ask price + slippage
`QuoteBar`	Sell	Best effort bid price - slippage

The model checks the data format in the following order:

1. `Tick`
2. `TradeBar`
3. `QuoteBar`

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Combo Market Orders

The fill logic of combo market orders depends on the data format of the security subscription and the order direction. The following table shows the fill price of combo market orders given these factors. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	Ask quote price + slippage
`Tick`	`Quote`	Sell	Bid quote price - slippage
`Tick`	`Trade`	Buy	Trade price + slippage
`Tick`	`Trade`	Sell	Trade price - slippage
`QuoteBar`		Buy	Ask close price + slippage
`QuoteBar`		Sell	Bid close price - slippage
`TradeBar`		Buy	Close price + slippage
`TradeBar`		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

The fill logic of combo limit orders depends on the data format of the security subscription and the order direction. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate

price. The following table shows how the fill model calculates the security prices.

Data Format	TickType	Combo Order Direction	Leg Order Direction	Price
Tick	Quote	Buy or sell	Buy	Ask price
Tick	Quote	Buy or sell	Sell	Bid price
Tick	Trade	Buy or sell	Buy or sell	Trade price
QuoteBar		Buy	Buy	Ask low price
QuoteBar		Buy	Sell	Bid low price
QuoteBar		Sell	Buy	Ask high price
QuoteBar		Sell	Sell	Bid high price
TradeBar		Buy	Buy or sell	Low price
TradeBar		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The fill logic of combo leg limit orders depends on the data format of the security subscription and the order direction. The following table shows the fill price of combo leg limit orders given these factors. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Future Model

Introduction

The `FutureFillModel` is the default fill model if you trade Futures contracts with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
security.SetFillModel(FutureFillModel())
```

PY

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

To view the implementation of this model, see the LEAN GitHub repository .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	`TickType`	Combo Order Direction	Leg Order Direction	Price
`Tick`	`Quote`	Buy or sell	Buy	Ask price
`Tick`	`Quote`	Buy or sell	Sell	Bid price
`Tick`	`Trade`	Buy or sell	Buy or sell	Trade price
`QuoteBar`		Buy	Buy	Ask low price
`QuoteBar`		Buy	Sell	Bid low price
`QuoteBar`		Sell	Buy	Ask high price
`QuoteBar`		Sell	Sell	Bid high price
`TradeBar`		Buy	Buy or sell	Low price
`TradeBar`		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Future Option Model

Introduction

The `FutureOptionFillModel` is the default fill model if you trade Future Option contracts with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
security.SetFillModel(FutureOptionFillModel())
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

To view the implementation of this model, see the LEAN GitHub repository .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	`TickType`	Combo Order Direction	Leg Order Direction	Price
`Tick`	`Quote`	Buy or sell	Buy	Ask price
`Tick`	`Quote`	Buy or sell	Sell	Bid price
`Tick`	`Trade`	Buy or sell	Buy or sell	Trade price
`QuoteBar`		Buy	Buy	Ask low price
`QuoteBar`		Buy	Sell	Bid low price
`QuoteBar`		Sell	Buy	Ask high price
`QuoteBar`		Sell	Sell	Bid high price
`TradeBar`		Buy	Buy or sell	Low price
`TradeBar`		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Immediate Model

Introduction

The `ImmediateFillModel` is the default fill model if you trade non-Equity assets with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
security.SetFillModel(ImmediateFillModel())
```
PY

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

To view the implementation of this model, see the LEAN GitHub repository .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	`TickType`	Combo Order Direction	Leg Order Direction	Price
`Tick`	`Quote`	Buy or sell	Buy	Ask price
`Tick`	`Quote`	Buy or sell	Sell	Bid price
`Tick`	`Trade`	Buy or sell	Buy or sell	Trade price
`QuoteBar`		Buy	Buy	Ask low price
`QuoteBar`		Buy	Sell	Bid low price
`QuoteBar`		Sell	Buy	Ask high price
`QuoteBar`		Sell	Sell	Bid high price
`TradeBar`		Buy	Buy or sell	Low price
`TradeBar`		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Latest Price Model

Introduction

The `LatestPriceFillModel` fills trades completely and immediately.

```
security.SetFillModel(LatestPriceFillModel())
```

PY

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If there is no `QuoteBar` available, it uses the OHLC prices from the `Security` object. If there is a `QuoteBar` available, the fill model gets the most recent `TradeBar` . If the `TradeBar` is more recent, it uses the `TradeBar` data. Otherwise, it uses the `QuoteBar` data.

To view the implementation of this model, see the LEAN GitHub repository .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	`TickType`	Combo Order Direction	Leg Order Direction	Price
`Tick`	`Quote`	Buy or sell	Buy	Ask price
`Tick`	`Quote`	Buy or sell	Sell	Bid price
`Tick`	`Trade`	Buy or sell	Buy or sell	Trade price
`QuoteBar`		Buy	Buy	Ask low price
`QuoteBar`		Buy	Sell	Bid low price
`QuoteBar`		Sell	Buy	Ask high price
`QuoteBar`		Sell	Sell	Bid high price
`TradeBar`		Buy	Buy or sell	Low price
`TradeBar`		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Reality Modeling

Slippage

Slippage

Key Concepts

Introduction

Slippage is the difference between the fill price you expect to get for an order and the actual fill price. Since the price can move in the direction of your trade or against the direction of your trade while you wait for the order to fill, slippage can be positive or negative. Slippage models model slippage to make backtest results more realistic.

Factors Impacting Slippage

There are many factors that can impact slippage, including the trading engine, brokerage connection, and market dynamics.

Trading Engine

How long does it take from when you place an order to when it's sent to the brokerage? Longer delays lead to more slippage.

Brokerage Connection

How long does it take for your brokerage to receive an order that you send? Slow internet connections, long travel distances, and poor infrastructure lead to more slippage

Market Dynamics

How volatile is the current market environment? More volatility leads to more slippage.

Does the market consist of sophisticated microsecond arbitrageurs? If your order creates an arbitrage opportunity, it can cause more slippage.

Set Models

The brokerage model of your algorithm automatically sets the slippage model for each security, but you can override it. To manually set the slippage model of a security, call the `SetSlippageModel` method on the `Security` object.

```py
# In Initialize
security = self.AddEquity("SPY")
security.SetSlippageModel(VolumeShareSlippageModel())
```

You can also set the slippage model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetSlippageModel(VolumeShareSlippageModel())
```

To view all the pre-built slippage models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the slippage model of each security. The default brokerage model is the `DefaultBrokerageModel` , which uses the ConstantSlippageModel to model zero slippage for all securities.

Model Structure

Slippage models should implement the `ISlippageModel` interface. Extensions of the `ISlippageModel` interface must implement the `GetSlippageApproximation` method, which calculates the slippage quantity.

```
PY
class MySlippageModel:

    def GetSlippageApproximation(self, asset: Security, order: Order) -> float:
        return asset.Price * 0.0001 * np.log10(2*float(order.AbsoluteQuantity))
```

Examples

Demonstration Algorithms
CustomModelsAlgorithm.py Python

Slippage

Supported Models

Introduction

This page describes the pre-built slippage models in LEAN. If none of these models perform exactly how you want, create a custom slippage model .

Constant Model

The `ConstantSlippageModel` applies a constant percentage of slippage to each order. It's the default slippage model of the DefaultBrokerageModel .

```
security.SetSlippageModel(ConstantSlippageModel(0))
```
PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`slippagePercent`	`float`	The slippage percent for each order. The value must be in the interval (0, 1).	

To view the implementation of this model, see the LEAN GitHub repository .

Volume Share Model

The `VolumeShareSlippageModel` calculates the slippage of each order by multiplying the price impact constant by the square of the ratio of the order to the total volume. If the volume of the current bar is zero, the slippage percent is

$$volumeLimit^2 * priceImpact$$

where **volumeLimit** and **priceImpact** are custom input variables. If the volume of the current bar is positive, the slippage percent is

$$\min(\frac{|orderQuantity|}{barVolume}, volumeLimit)^2 * priceImpact$$

where **orderQuantity** is the quantity of the order and **barVolume** is the volume of the current bar. If the security subscription provides `TradeBar` data, the **barVolume** is the volume of the current bar. If the security subscription provides `QuoteBar` data, the **barVolume** is the bid or ask size of the current bar. CFD, Forex, and Crypto data often doesn't include volume. If there is no volume reported for these asset classes, the model returns zero slippage.

```
security.SetSlippageModel(VolumeShareSlippageModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`volumeLimit`	`float`	Maximum percent of historical volume that can fill in each bar. 0.5 means 50% of historical volume. 1.0 means 100%.	0.025
`priceImpact`	`float`	Scaling coefficient for price impact. Larger values will result in more simulated price impact. Smaller values will result in less simulated price impact.	0.1

To view the implementation of this model, see the LEAN GitHub repository .

Reality Modeling

Transaction Fees

Transaction Fees

Key Concepts

Introduction

Your orders incur a transaction fee when a brokerage fills them in the market. LEAN uses transaction fee models in backtesting to model the live trading fees you would incur with the strategy. Transaction fee models make backtest results more realistic. To give your backtests the most accurate fees, LEAN contains transaction fee models that model the fee structure of many popular brokerages.

Set Models

The brokerage model of your algorithm automatically sets the fee model for each security, but you can override it. To manually set the fee model of a security, call the `SetFeeModel` method on the `Security` object.

PY

```
# In Initialize
security = self.AddEquity("SPY")
security.SetFeeModel(ConstantFeeModel(0))
```

You can also set the fee model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

PY

```
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetFeeModel(ConstantFeeModel(0))
```

In live trading, the `SetFeeModel` method isn't ignored. If we use order helper methods like `SetHoldings` , the fee model helps to calculate the order quantity. However, the algorithm doesn't update the cash book with the fee from the fee model. The algorithm uses the actual fee from the brokerage to update the cash book.

To view all the pre-built fee models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the fill model for each security. The default brokerage model is the `DefaultBrokerageModel` , which set the ConstantFeeModel with no fees for Forex, CFD, and Crypto assets and sets the InteractiveBrokersFeeModel for the remaining asset classes.

Model Structure

Fee models should extend the `FeeModel` class. Extensions of the `FeeModel` class must implement the `GetOrderFee` method, which receives `OrderFeeParameters` and returns an `OrderFee` that represents a cash amount in a currency.

```
PY
class MyFeeModel(FeeModel):

    def GetOrderFee(self, parameters: OrderFeeParameters) -> OrderFee:
        return OrderFee(CashAmount(0.5, 'USD'))
```

The `OrderFeeParameters` object has the following members:

Negative Transaction Fees

If you short a security and receive interest payments, they are negative transaction fees.

Examples

Demonstration Algorithms
CustomModelsAlgorithm.py Python RawPricesCoarseUniverseAlgorithm.py Python

Transaction Fees

Supported Models

Introduction

This page describes some of the pre-built fee models in LEAN. For more brokerage-specific fee models, see the brokerage model documentation . If none of these models perform exactly how you want, create a custom fee model .

Constant Model

The `ConstantFeeModel` applies the absolute value of a constant fee to each order. It's the default fee model of the DefaultBrokerageModel if you trade Forex, CFD, or Crypto assets.

```
security.SetFeeModel(ConstantFeeModel(0.05))
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fee`	`float`	The order fee quantity	
`currency`	`str`	The order fee currency	"USD"

To view the implementation of this model, see the LEAN GitHub repository .

Interactive Brokers Model

The `InteractiveBrokersFeeModel` models the fees of Interactive Brokers .

```
security.SetFeeModel(InteractiveBrokersFeeModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`monthlyForexTradeAmountInUSDollars`	`float`	Monthly Forex dollar volume traded	0
`monthlyOptionsTradeAmountInContracts`	`float`	Monthly Option contracts traded	0

The following table describes which currency the `InteractiveBrokersFeeModel` charges fees in for each asset class:

Asset Class	Fee Currency
US Equity	USD
India Equity	INR
Equity Options	USD
Forex	USD
US Futures	USD
Hong Kong Futures	The contract quote currency (CNH, HKD, or USD)
US Future Options	USD
Hong Kong Future Options	The contract quote currency (CNH, HKD, or USD)
Index Options	USD

The following sections describe the trading fees of each asset class. To view the implementation of this model, see the LEAN GitHub repository .

US Equities

US Equity trades cost 0.005/share with a 1 minimum fee and a 0.5% maximum fee.

Equity Options and Index Options

Equity Options and Index Options fees are a function of your monthly volume and the premium of the contract you trade. The following table shows the fees for each volume tier:

Monthly Volume (Contacts)	Premium ($)	Fee per Contract ($)
<= 10,000	< 0.05	0.25
<= 10,000	0.05 <= premium < 0.10	0.50
<= 10,000	>= 0.10	0.70
10,000 < volume <= 50,000	< 0.05	0.25
10,000 < volume <= 50,000	>= 0.05	0.50
50,000 < volume <= 100,000	Any	0.25
> 100,000	Any	0.15

By default, LEAN models your fees at the tier with the lowest monthly volume. To adjust the fee tier, manually set the fee model and provide a `monthlyOptionsTradeAmountInContracts` argument.

There is no fee to exercise Option contracts.

Forex

Forex fees are a function of your monthly Forex trading volume. The following table shows the fee tiers:

Monthly Volume (USD)	Commission Rate (%)	Minimum Fee ($)
<= 1B	0.002	2
1B < volume <= 2B	0.0015	1.5
2B < volume <= 5B	0.001	1.25
> 5B	0.0008	1

By default, LEAN models your fees at the tier with the lowest monthly volume. To adjust the fee tier, manually set the fee model and provide a `monthlyForexTradeAmountInUSDollars` argument.

US Futures

US Futures fees depend on the contracts you trade. The following table shows the base fee per contract for each Future:

Contract Symbol	Market	Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
E-mini Futures				
ES	CME	E-mini S&P 500 Futures	0.85	1.28
NQ	CME	E-mini Nasdaq-100 Futures	0.85	1.28
YM	CBOT	E-mini Dow ($5) Futures	0.85	1.28
RTY	CME	E-mini Russell 2000 Index Futures	0.85	1.28
EMD	CME	E-mini S&P MidCap 400 Futures	0.85	1.28
Micro E-mini Futures				
MYM	CBOT	Micro E-mini Dow Jones Industrial Average Index Futures	0.25	0.3
M2K	CME	Micro E-mini Russell 2000 Index Futures	0.25	0.3

Contract Symbol	Market	Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
MES	CME	Micro E-mini Standard and Poor's 500 Stock Price Index Futures	0.25	0.3
MNQ	CME	Micro E-mini Nasdaq-100 Index Futures	0.25	0.3
2YY	CBOT	Micro 2-Year Yield Futures	0.25	0.3
5YY	CBOT	Micro 5-Year Yield Futures	0.25	0.3
10Y	CBOT	Micro 10-Year Yield Futures	0.25	0.3
30Y	CBOT	Micro 30-Year Yield Futures	0.25	0.3
MCL	NYMEX	Micro WTI Crude Oil Futures	0.25	0.3
MGC	COMEX	Micro Gold Futures	0.25	0.3
SIL	COMEX	Micro Silver Futures	0.25	0.3
Cryptocurrency Futures				
BTC	CME	Bitcoin Futures	5	6
MIB	CME	BTIC on Micro Bitcoin Futures	2.25	2.5
MBT	CME	Micro Bitcoin Futures	2.25	2.5
MET	CME	Micro Ether Futures	0.2	0.2
MRB	CME	BTIC on Micro Ether Futures	0.2	0.2
E-mini FX Futures				
E7	CME	E-mini Euro FX Futures	0.5	0.85
J7	CME	E-mini Japanese Yen Futures	0.5	0.85
Micro E-mini FX Futures				

Contract Symbol	Market	Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
M6E	CME	Micro Euro/U.S. Dollar (EUR/USD) Futures	0.15	0.24
M6A	CME	Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures	0.15	0.24
M6B	CME	Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures	0.15	0.24
MCD	CME	Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures	0.15	0.24
MJY	CME	Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures	0.15	0.24
MSF	CME	Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures	0.15	0.24
M6J	CME	Micro USD/JPY Futures	0.15	0.24
MIR	CME	Micro INR/USD Futures	0.15	0.24
M6C	CME	Micro USD/CAD Futures	0.15	0.24
M6S	CME	Micro USD/CHF Futures	0.15	0.24
MNH	CME	Micro USD/CNH Futures	0.15	0.24

If you trade a contract that's not in the preceding table, the base fee is 0.85/contract and the exchange fee is 1.60/contract.

In addition to the base fee and exchange fee, there is a $0.02/contract regulatory fee.

Futures Options

Futures Options fees depend on the contracts you trade. The following table shows the base fee per contract for each Future:

Contract Symbol	Market	Underlying Futures Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
E-mini Futures Options				

Contract Symbol	Market	Underlying Futures Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
ES	CME	E-mini S&P 500 Futures	0.85	0.55
NQ	CME	E-mini Nasdaq-100 Futures	0.85	0.55
YM	CBOT	E-mini Dow ($5) Futures	0.85	0.55
RTY	CME	E-mini Russell 2000 Index Futures	0.85	0.55
EMD	CME	E-mini S&P MidCap 400 Futures	0.85	0.55
Micro E-mini Futures Options				
MYM	CBOT	Micro E-mini Dow Jones Industrial Average Index Futures	0.25	0.2
M2K	CME	Micro E-mini Russell 2000 Index Futures	0.25	0.2
MES	CME	Micro E-mini Standard and Poor's 500 Stock Price Index Futures	0.25	0.2
MNQ	CME	Micro E-mini Nasdaq-100 Index Futures	0.25	0.2
2YY	CBOT	Micro 2-Year Yield Futures	0.25	0.2
5YY	CBOT	Micro 5-Year Yield Futures	0.25	0.2
10Y	CBOT	Micro 10-Year Yield Futures	0.25	0.2
30Y	CBOT	Micro 30-Year Yield Futures	0.25	0.2
MCL	NYMEX	Micro WTI Crude Oil Futures	0.25	0.2
MGC	COMEX	Micro Gold Futures	0.25	0.2
SIL	COMEX	Micro Silver Futures	0.25	0.2

Contract Symbol	Market	Underlying Futures Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
Cryptocurrency Futures Options				
BTC	CME	Bitcoin Futures	5	5
MIB	CME	BTIC on Micro Bitcoin Futures	1.25	2.5
MBT	CME	Micro Bitcoin Futures	1.25	2.5
MET	CME	Micro Ether Futures	0.1	0.2
MRB	CME	BTIC on Micro Ether Futures	0.1	0.2

If you trade a contract that's not in the preceding table, the base fee is 0.85/contract and the exchange fee is 1.60/contract.

In addition to the base fee and exchange fee, there is a $0.02/contract regulatory fee.

Binance Model

The `BinanceFeeModel` models the fees of spot trades on Binance and Binance US .

```
security.SetFeeModel(BinanceFeeModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`mFee`	`float`	Maker fee value	0.001
`tFee`	`float`	Taker fee value	0.001

The `BinanceFeeModel` charges the order fees of Binance and Binance US at the VIP 0 level, which is a 0.1% maker and taker fee. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Binance adjusts your fees based on your 30-day trading volume and BNB balance, but we don't currently model these metrics to adjust fees.

The `BinanceFeeModel` charges fees in the currency you receive from a trade. For example, if you buy ETHBTC, you pay fees in ETH. If you sell ETHBTC, you pay fees in BTC.

To view the implementation of this model, see the LEAN GitHub repository .

Binance Futures Model

The `BinanceFuturesFeeModel` models the Binance Futures live trading fees .

```
security.SetFeeModel(BinanceFuturesFeeModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`mUsdtFee`	`float`	Maker fee value for USDT pair contracts	0.0002
`tUsdtFee`	`float`	Taker fee value for USDT pair contracts	0.0004
`mBusdFee`	`float`	Maker fee value for BUSD pair contracts	0.00012
`tUsdtFee`	`float`	Taker fee value for BUSD pair contracts	0.00036

By default, the `BinanceFuturesFeeModel` charges the order fees of Binance Futures at the Regular User level. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Binance adjusts your fees based on your 30-day trading volume and BNB balance, but we don't currently model these metrics to adjust fees.

The `BinanceFuturesFeeModel` charges fees in the currency you receive from a trade. For example, if you buy BTCUSD, you pay fees in BTC. If you sell BTCUSD, you pay fees in USD.

To view the implementation of this model, see the LEAN GitHub repository .

Binance Coin Futures Model

The `BinanceCoinFuturesFeeModel` models the Binance Coin Futures live trading fees .

```
security.SetFeeModel(BinanceCoinFuturesFeeModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`mFee`	`float`	Maker fee value	0.0001
`tFee`	`float`	Taker fee value	0.0005

By default, the `BinanceCoinFuturesFeeModel` charges the order fees of Binance Coin Futures at the Regular User level. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Binance adjusts your fees based on your 30-day trading volume and BNB balance, but we don't currently model these metrics to adjust fees.

The `BinanceCoinFuturesFeeModel` charges fees in the currency you receive from a trade. For example, if you buy BTCUSDT, you pay fees in BTC. If you sell BTCUSDT, you pay fees in USDT.

To view the implementation of this model, see the LEAN GitHub repository .

Bitfinex Model

The `BitfinexFeeModel` models the Bitfinex live trading fees .

```
security.SetFeeModel(BitfinexFeeModel())
```

The `BitfinexFeeModel` charges a 0.1% maker fee and a 0.2% taker fee. If you place a limit order that hits a hidden order or you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you place a hidden order or you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Bitfinex adjusts your fees based on your 30-day trading volume and LEO balance, but we don't currently model these metrics to adjust fees.

The `BitfinexFeeModel` charges fees in the currency you receive from a trade. For example, if you buy ETHBTC, you pay fees in ETH. If you sell ETHBTC, you pay fees in BTC.

To view the implementation of this model, see the LEAN GitHub repository .

GDAX Model

The `GDAXFeeModel` models the Coinbase live trading fees .

```
security.SetFeeModel(GDAXFeeModel())
```

The `GDAXFeeModel` models the order fees of Coinbase at the $50K-100K pricing tier for all Crypto pairs, so it charges a 0.5% maker and taker fee for most pairs. The following table shows the Coinbase Stable Pairs, which charge a 0% maker fee and a 0.1% taker fee:

Stable Pairs			
DAIUSDC	DAIUSD	GYENUSD	PAXUSD
PAXUSDT	MUSDUSD	USDCEUR	USDCGBP
USDTEUR	USDTGBP	USDTUSD	USDTUSDC
USTEUR	USTUSD	USTUSDT	WBTCBTC

If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Coinbase adjusts your fees based on your 30-day trading volume, but we don't currently model trading volume to adjust fees.

The `GDAXFeeModel` models the adjustments Coinbase has made to their fees over time. The following table shows the fees for each time period:

Time Period (UTC)	Maker Fee (%)	Taker Fee (%)
Time < 3/23/2019 1:30AM	0	0.3
3/23/2019 1:30AM <= Time < 10/8/2019 12:30AM	0.15	0.25
10/8/2019 12:30AM <= Time	0.5	0.5

The `GDAXFeeModel` charges fees in the quote currency .

To view the implementation of this model, see the LEAN GitHub repository .

Kraken Model

The `KrakenFeeModel` models the fees of Kraken .

```
security.SetFeeModel(KrakenFeeModel())
```

The `KrakenFeeModel` model the order fees of Kraken. For trading pairs that contain only Crypto assets, this fee model models the lowest tier in Kraken's tiered fee structure, which is a 0.16% maker fee and a 0.26% taker fee. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. For trading pairs that have any of the following currencies as the base currency in the pair, the fee is 0.2%:

- CAD
- EUR
- GBP
- JPY
- USD
- USDT
- DAI
- USDC

The `KrakenFeeModel` charges fees in the quote currency for buy orders and in the base currency for sell orders. To override these settings, define the `KrakenOrderProperties` .

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = KrakenOrderProperties()
    self.DefaultOrderProperties.FeeInBase = True
    self.DefaultOrderProperties.FeeInQuote = False

def OnData(self, slice: Slice) -> None:
    # Override the default order properties
    order_properties = KrakenOrderProperties()
    order_properties.FeeInQuote = True
    order_properties.FeeInBase = False
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Kraken adjusts your fees based on your 30-day trading volume, but the `KrakenFeeModel` don't currently model trading volume to adjust fees.

To view the implementation of this model, see the LEAN GitHub repository .

Samco Model

The `SamcoFeeModel` models the fees of Samco .

```
security.SetFeeModel(SamcoFeeModel())
```

The `SamcoFeeModel` models the order fees of Samco by its Equity Intraday fee structure. The following table shows the fees:

Charge Item	Fee
Brokerage Fee	₹20 per trade or 0.02% (whichever is lower)
Exchange Transaction Charge	0.00345%
Securities Transaction Tax	0.025%
Goods and Services Tax	18%
SEBI Charges	0.0001%
Stamp Duty	0.003%

The `SamcoFeeModel` charges fees in INR.

To view the implementation of this model, see the LEAN GitHub repository .

TD Ameritrade Model

The `TDAmeritradeFeeModel` models the fees of TD Ameritrade .

```
security.SetFeeModel(TDAmeritradeFeeModel())
```

The `TDAmeritradeFeeModel` charges $0 USD for Equity trades.

To view the implementation of this model, see the LEAN GitHub repository .

Wolverine Model

The `WolverineFeeModel` models the fees of Wolverine Execution Services.

```
security.SetFeeModel(WolverineFeeModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`feesPerShare`	`float` / `NoneType`	The fees per share to apply. If `None` , it uses 0.005.	`None`

The `WolverineFeeModel` charges fees in USD.

To view the implementation of this model, see the LEAN GitHub repository .

Zerodha Model

The `ZerodhaFeeModel` models the fees of Zerodha .

```
security.SetFeeModel(ZerodhaFeeModel())
```

PY

The `ZerodhaFeeModel` models the order fees of Zerodha with its Equity Intraday fee structure. The following table shows the fees:

Charge Item	Fee
Brokerage Fee	₹20 per trade or 0.03% (whichever is lower)
Exchange Transaction Charge	0.00345%
Securities Transaction Tax	0.025%
Goods and Services Tax	18%
SEBI Charges	0.0001%
Stamp Duty	0.003%

The `ZerodhaFeeModel` charges fees in INR.

To view the implementation of this model, see the LEAN GitHub repository .

Reality Modeling

Brokerages

Brokerages

Key Concepts

Introduction

Brokerages provide you with a connection to the market so you can fill trades. Brokerage models simulate the live behavior of a real brokerage. To avoid sending invalid orders for execution in live trading, the brokerage model validates your orders before LEAN sends them to the real brokerage. Brokerage models combine together all of the models relevant for a brokerage. If you set the appropriate brokerage model, the supported order types, default markets, and some of the security-level models are appropriately set in your algorithm.

Set Models

To set a brokerage model, in the `Initialize` method, call the `SetBrokerageModel` method with a `BrokerageName` and an `AccountType` . If you set a brokerage model, it overrides any security level models you manually set in your algorithm.

```
PY
self.SetBrokerageModel(BrokerageName.OandaBrokerage) # Defaults to margin account
self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin) # Overrides the default account type
```

In live trading, LEAN doesn't ignore your `SetBrokerageModel` method calls. LEAN uses some of the brokerage model rules to catch invalid orders before they reach your real brokerage.

To view all the pre-built brokerage models, see Supported Models .

Default Behavior

The default brokerage model is the `DefaultBrokerageModel` , but LEAN has many other brokerage models you can use in your algorithms. For more information about the `DefaultBrokerageModel` , see QuantConnect Paper Trading .

Modeled Properties

Brokerage models model that following properties:

- Transaction fees
- Supported asset classes
- Account types
- Support for extended market hours

- Leverage for assets
- Trade settlement rules
- Order properties and updates

Model Structure

Brokerage models should extend the `DefaultBrokerageModel` class. Extensions of the `DefaultBrokerageModel` class should implement the following methods:

```
class MyBrokerageModel(DefaultBrokerageModel):
    DefaultMarkets = {}
    RequiredFreeBuyingPowerPercent = 0

    def __init__(self, accountType: AccountType = AccountType.Margin):
        self.AccountType = accountType
        self.ShortableProvider = NullShortableProvider()

    def CanSubmitOrder(self, security: Security, order: Order,
         message: BrokerageMessageEvent) -> bool:
        return super().CanSubmitOrder(security, order, message)

    def CanUpdateOrder(self, security: Security, order: Order,
         request: UpdateOrderRequest, message: BrokerageMessageEvent) -> bool:
        return super().CanUpdateOrder(security, order, request, message)

    def CanExecuteOrder(self, security: Security, order: Order) -> bool:
        return super().CanExecuteOrder(security, order)

    def ApplySplit(self, tickets: List[OrderTicket], split: Split) -> None:
        super().ApplySplit(tickets, split)

    def GetLeverage(self, security: Security) -> float:
        return super().GetLeverage(security)

    def GetBenchmark(self, securities: SecurityManager) -> IBenchmark:
        return super().GetBenchmark(securities)

    def GetFillModel(self, security: Security) -> IFillModel:
        return super().GetFillModel(security)

    def GetFeeModel(self, security: Security) -> IFeeModel:
        return super().GetFeeModel(security)

    def GetSlippageModel(self, security: Security) -> ISlippageModel:
        return super().GetSlippageModel(security)

    def GetSettlementModel(self, security: Security) -> ISettlementModel:
        return super().GetSettlementModel(security)

    def GetBuyingPowerModel(self, security: Security) -> IBuyingPowerModel:
        return super().GetBuyingPowerModel(security)

    def GetMarginInterestRateModel(self, security: Security) -> IMarginInterestRateModel:
        return super().GetMarginInterestRateModel(security)

    def GetShortableProvider(self) -> IShortableProvider:
        return self.ShortableProvider
```

Custom brokerage models give you enormous control over your algorithm behavior and allow you to model virtually any brokerage in the world.

Examples

Demonstration Algorithms
BrokerageModelAlgorithm.py Python

Brokerages

Supported Models

Brokerage models combine together all of the models relevant for a brokerage. The following brokerage models are available:

QuantConnect Paper Trading

Binance

Bitfinex

Coinbase

Interactive Brokers

Kraken

Oanda

Samco

TD Ameritrade

Tradier

Trading Technologies

Wolverine

Zerodha

See Also

Reality Modeling
Live Trading Brokerages

Supported Models

QuantConnect Paper Trading

Introduction

This page explains the `DefaultBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage) # Defaults to margin account
self.SetBrokerageModel(BrokerageName.QuantConnectBrokerage, AccountType.Margin) # Overrides the default
account type
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `DefaultBrokerageModel` supports trading for all asset classes .

Orders

The following sections describe how the `DefaultBrokerageModel` handles orders.

Order Types

The following table describes the available order types for each asset class that the `DefaultBrokerageModel` supports:

Order Type	US Equity	Equity Options	Crypto	Crypto Futures	Forex	CFD	Futures	Futures Options	Index Options
MarketOrder	✓	✓	✓	✓	✓	✓	✓	✓	✓
LimitOrder	✓	✓	✓	✓	✓	✓	✓	✓	✓
LimitIfTouchedOrder	✓	✓	✓	✓	✓	✓	✓	✓	✓
StopMarketOrder	✓	✓	✓	✓	✓	✓	✓	✓	✓
StopLimitOrder	✓	✓	✓	✓	✓	✓	✓	✓	✓
MarketOnOpenOrder	✓	✓	✓		✓	✓			✓
MarketOnCloseOrder	✓	✓	✓		✓	✓	✓	✓	✓
ComboMarketOrder		✓						✓	✓
ComboLimitOrder		✓						✓	✓
ComboLegLimitOrder		✓						✓	✓
ExerciseOption		✓ Not supported for cash-settled Options						✓	

Time In Force

The `DefaultBrokerageModel` supports the following TimeInForce instructions:

- `Day`
- `GoodTilCanceled`
- `GoodTilDate`

Updates

The `DefaultBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The following table shows the fill model that the `DefaultBrokerageModel` uses for each `SecurityType` :

`SecurityType`	**Fill Model**
`Equity`	EquityFillModel
`Future`	FutureFillModel
`FutureOption`	FutureOptionFillModel
Remaining `SecurityType` values	ImmediateFillModel

Slippage

The `DefaultBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
PY

security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `DefaultBrokerageModel` uses the ConstantFeeModel with no fees for Forex, CFD, Crypto, and Crypto Future assets and the InteractiveBrokersFeeModel for the remaining asset classes.

```
PY

# For Forex, CFD, Crypto, and Crypto Future assets:
security.SetFeeModel(ConstantFeeModel(0))

# For the remaining asset classes:
security.SetFeeModel(InteractiveBrokersFeeModel())
```

Buying Power

The `DefaultBrokerageModel` sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	Model
Equity Options	`OptionMarginModel`
Futures	`FutureMarginModel`
Future Options	`FuturesOptionsMarginModel`
Index Options	`OptionMarginModel`
Crypto	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
CryptoFuture	`CryptoFutureMarginModel`
Forex	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
Other	`SecurityMarginModel`

If you have a margin account, the `DefaultBrokerageModel` allows 2x leverage for Equities, 50x leverage for Forex and CFDs, and 1x leverage for the remaining asset classes.

Settlement

The following table shows which settlement model the `DefaultBrokerageModel` uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `DefaultBrokerageModel` uses the ImmediateSettlementModel .

```py
# For US Equities with a cash account:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For Equity Options with a cash account:
security.SettlementModel = DelayedSettlementModel(Option.DefaultSettlementDays,
Option.DefaultSettlementTime)

# For Futures
security.SettlementModel = FutureSettlementModel()

# For remaining cases:
security.SettlementModel = ImmediateSettlementModel()
```

Margin Interest Rate

The `DefaultBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `DefaultBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME
FutureOption	CME
Index	USA
IndexOption	USA
Forex	Oanda
Cfd	Oanda
Crypto	GDAX
CryptoFuture	Binance

Supported Models

Binance

Introduction

This page explains the Binance brokerage models, including the asset classes they supports, their default security-level models , and their default markets.

```
# Binance Spot
self.SetBrokerageModel(BrokerageName.Binance, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Binance, AccountType.Margin)

# Binance Futures
self.SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin)
self.SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin)

# Binance US Spot
self.SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash)
```

To view the implementation of these models, see the following pages in the LEAN GitHub repository:

- BinanceBrokerageModel.cs
- BinanceFuturesBrokerageModel.cs
- BinanceCoinFuturesBrokerageModel.cs
- BinanceUSBrokerageModel.cs

Asset Classes

The Binance brokerage models support trading Crypto and Crypto Futures .

The Binance US brokerage model supports trading Crypto .

Orders

The Binance and Binance US brokerage models support several order types and order properties, but don't support order updates.

Order Types

The following table describes the available order types for each asset class that the Binance and Binance US brokerage models support:

Order Type	Crypto	Crypto Futures
MarketOrder	✓	✓
LimitOrder	✓	✓
StopLimitOrder	✓	

Order Properties

The Binance and Binance US brokerage models supports custom order properties. The following table describes the members of the `BinanceOrderProperties` object that you can set to customize order execution:

Property	Description
`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`
`PostOnly`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.

PY

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = BinanceOrderProperties()
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled
    self.DefaultOrderProperties.PostOnly = False

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = BinanceOrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    order_properties.PostOnly = True
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(year, month, day))
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The Binance and Binance US brokerage models don't support order updates, but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The Binance brokerage models use the ImmediateFillModel .

Slippage

The Binance brokerage models use the ConstantSlippageModel with zero slippage.

PY

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The following table shows the fee model that each Binance brokerage model uses:

Brokerage Model	Fee Model
`BinanceBrokerageModel`	BinanceFeeModel
`BinanceFuturesBrokerageModel`	BinanceFuturesFeeModel
`BinanceCoinFuturesBrokerageModel`	BinanceCoinFuturesFeeModel
`BinanceUSBrokerageModel`	BinanceFeeModel

The Binance brokerage models use the default argument values for each fee model so that it models the current Binance fee schedule.

Buying Power

The Binance brokerage models use the `CashBuyingPowerModel` for cash accounts and the `SecurityMarginModel` for margin accounts.

The following table shows the maximum leverage each brokerage model allows for margin accounts:

Brokerage Model	Maximum Leverage
`BinanceBrokerageModel`	3
`BinanceFuturesBrokerageModel`	25
`BinanceCoinFuturesBrokerageModel`	25

The `BinanceUSBrokerageModel` doesn't currently support margin trading.

Settlement

The Binance brokerage models use the ImmediateSettlementModel .

Margin Interest Rate

The following table shows the margin interest rate model that the Binance brokerages use:

Brokerage Model	Margin Interest Rate Model
`BinanceBrokerageModel`	NullMarginInterestRateModel
`BinanceFuturesBrokerageModel`	BinanceFutureMarginInterestRateModel for Crypto Perpetual Futures and NullMarginInterestRateModel for other assets
`BinanceCoinFuturesBrokerageModel`	BinanceFutureMarginInterestRateModel for Crypto Perpetual Futures and NullMarginInterestRateModel for other assets
`BinanceUSBrokerageModel`	NullMarginInterestRateModel

Default Markets

The following table shows the default market of the Binance brokerage models:

Brokerage Model	Market
`BinanceBrokerageModel`	`Market.Binance`
`BinanceFuturesBrokerageModel`	`Market.Binance`
`BinanceCoinFuturesBrokerageModel`	`Market.Binance`
`BinanceUSBrokerageModel`	`Market.BinanceUS`

Supported Models

Bitfinex

Introduction

This page explains `BitfinexBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `BitfinexBrokerageModel` supports trading Crypto .

Orders

The `BitfinexBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `BitfinexBrokerageModel` supports:

Order Type	Crypto
MarketOrder	✓
LimitOrder	✓
LimitIfTouchedOrder	✓
StopMarketOrder	✓
StopLimitOrder	✓
MarketOnOpenOrder	✓
MarketOnCloseOrder	✓

Order Properties

The `BitfinexBrokerageModel` supports custom order properties. The following table describes the members of the `BitfinexOrderProperties` object that you can set to customize order execution:

Property	Description
`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`
`Hidden`	A flag to signal that the order should be hidden. Hidden orders do not appear in the order book, so they do not influence other market participants. Hidden orders incur the taker fee.
`PostOnly`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.

PY

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = BitfinexOrderProperties()
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled
    self.DefaultOrderProperties.Hidden = False
    self.DefaultOrderProperties.PostOnly = False

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = BitfinexOrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    order_properties.Hidden = True
    order_properties.PostOnly = False
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(year, month, day))
    order_properties.Hidden = False
    order_properties.PostOnly = True
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `BitfinexBrokerageModel` supports order updates .

Fills

The `BitfinexBrokerageModel` uses the ImmediateFillModel .

Slippage

The `BitfinexBrokerageModel` uses the ConstantSlippageModel with zero slippage.

PY

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `BitfinexBrokerageModel` uses the BitfinexFeeModel with the default argument values that models the current

Bitfinex fee schedule.

Buying Power

The `BitfinexBrokerageModel` uses the `CashBuyingPowerModel` for cash accounts and the `SecurityMarginModel` for margin accounts.

If you have a margin account, the `BitfinexBrokerageModel` allows 3.3x leverage.

Settlement

The `BitfinexBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `BitfinexBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `BitfinexBrokerageModel` is `Market.Bitfinex` .

Supported Models

Coinbase

Introduction

This page explains `GDAXBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.GDAX, AccountType.Cash)
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `GDAXBrokerageModel` supports trading Crypto .

Orders

The `GDAXBrokerageModel` supports several order types and order properties, but it doesn't support order updates.

Order Types

The following table describes the available order types for each asset class that the `GDAXBrokerageModel` supports:

Order Type	Crypto
MarketOrder	✓
LimitOrder	✓
StopMarketOrder	✓ Supported after 2019-03-23 in backtests. For reference, see the Coinbase Market Structure Update on the Coinbase website.
StopLimitOrder	✓

Order Properties

The `GDAXBrokerageModel` supports custom order properties. The following table describes the members of the `GDAXOrderProperties` object that you can set to customize order execution:

Property	Description
`TimeInForce`	A TimeInForce instruction to apply to the order. The `GoodTilCanceled` `TimeInForce` is supported.
`PostOnly`	A flag that signals the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.

```
PY
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = GDAXOrderProperties()
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled
    self.DefaultOrderProperties.PostOnly = False

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = GDAXOrderProperties()
    order_properties.TimeInForce = TimeInForce.GoodTilCanceled
    order_properties.PostOnly = True
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `GDAXBrokerageModel` doesn't support order updates, but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The `GDAXBrokerageModel` uses the ImmediateFillModel .

Slippage

The `GDAXBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
PY
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `GDAXBrokerageModel` uses the GDAXFeeModel .

Buying Power

The `GDAXBrokerageModel` uses the `CashBuyingPowerModel` . The brokerage model doesn't currently support margin trading.

Settlement

The `GDAXBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `GDAXBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `GDAXBrokerageModel` is `Market.GDAX` .

Supported Models

Interactive Brokers

Introduction

This page explains `InteractiveBrokersBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `InteractiveBrokersBrokerageModel` supports the following asset classes:

- US Equities
- Equity Options
- Forex
- Futures
- Future Options
- Indices
- Index Options

Orders

The `InteractiveBrokersBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the order types that the `InteractiveBrokersBrokerageModel` supports: supports. For specific details about each order type, refer to the IB documentation.

Order Type	IB Documentation Page
MarketOrder	Market Orders
LimitOrder	Limit Orders
LimitIfTouchedOrder	Limit if Touched Orders
StopMarketOrder	Stop Orders
StopLimitOrder	Stop-Limit Orders
MarketOnOpenOrder	Market-on-Open (MOO) Orders
MarketOnCloseOrder	Market-on-Close (MOC) Orders
ComboMarketOrder	Spread Orders
ComboLimitOrder	Spread Orders
ComboLegLimitOrder	Spread Orders
ExerciseOption	Options Exercise

The following table describes the available order types for each asset class that the `InteractiveBrokersBrokerageModel` supports:

Order Type	US Equity	Equity Options	Forex	Futures	Futures Options	Index Options
MarketOrder	✓	✓	✓	✓	✓	✓
LimitOrder	✓	✓	✓	✓	✓	✓
LimitIfTouchedOrder	✓	✓	✓	✓	✓	✓
StopMarketOrder	✓	✓	✓	✓	✓	✓
StopLimitOrder	✓	✓	✓	✓	✓	✓
MarketOnOpenOrder	✓	✓	✓			✓
MarketOnCloseOrder	✓	✓	✓	✓	✓	✓
ComboMarketOrder		✓			✓	✓
ComboLimitOrder		✓			✓	✓
ComboLegLimitOrder		✓			✓	✓
ExerciseOption		✓ Not supported for cash-settled Options			✓	

Order Properties

The `InteractiveBrokersBrokerageModel` supports custom order properties. The following table describes the members of the `InteractiveBrokersOrderProperties` object that you can set to customize order execution. The table does not include the preceding methods for FA accounts.

Property	Description
`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`
`OutsideRegularTradingHours`	A flag to signal that the order may be triggered and filled outside of regular trading hours.

```py
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled
    self.DefaultOrderProperties.OutsideRegularTradingHours = False

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = InteractiveBrokersOrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    order_properties.OutsideRegularTradingHours = True
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(year, month, day))
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `InteractiveBrokersBrokerageModel` supports order updates .

Financial Advisor Group Orders

To place FA group orders, see Financial Advisors .

Fractional Trading

The `InteractiveBrokersBrokerageModel` doesn't support fractional trading .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Order Size Limits

The `InteractiveBrokersBrokerageModel` enforces the Spot Currency Minimum/Maximum Order Sizes from the IB website.

Fills

The following table shows the fill model that the `InteractiveBrokersBrokerageModel` uses for each `SecurityType` :

SecurityType	Fill Model
Equity	EquityFillModel
Future	FutureFillModel
FutureOption	FutureOptionFillModel
Remaining SecurityType values	ImmediateFillModel

Slippage

The InteractiveBrokersBrokerageModel uses the ConstantSlippageModel with zero slippage.

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The InteractiveBrokersBrokerageModel uses the InteractiveBrokersFeeModel with the default argument values. We model current Interactive Brokers fees on all assets.

Buying Power

The InteractiveBrokersBrokerageModel sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	Model
Equity Options	OptionMarginModel
Futures	FutureMarginModel
Future Options	FuturesOptionsMarginModel
Index Options	OptionMarginModel
Crypto	CashBuyingPowerModel for cash accounts or SecurityMarginModel for margin accounts
CryptoFuture	CryptoFutureMarginModel
Forex	CashBuyingPowerModel for cash accounts or SecurityMarginModel for margin accounts
Other	SecurityMarginModel

If you have a margin account, the InteractiveBrokersBrokerageModel allows 2x leverage for Equities, 50x leverage for Forex, and 1x leverage for the remaining asset classes.

Settlement

The `InteractiveBrokersBrokerageModel` uses the ImmediateSettlementModel in most cases. If you trade US Equities or Equity Options with a cash account, it uses the DelayedSettlementModel with the default settlement rules .

The following table shows which settlement model the `InteractiveBrokersBrokerageModel` uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `InteractiveBrokersBrokerageModel` uses the ImmediateSettlementModel .

```py
# For US Equities with a cash account:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For Equity Options with a cash account:
security.SettlementModel = DelayedSettlementModel(Option.DefaultSettlementDays,
Option.DefaultSettlementTime)

# For remaining cases:
security.SettlementModel = ImmediateSettlementModel()
```

Interactive Brokers doesn't provide information on which assets aren't settled, so we assume each live trading session starts with its cash fully settled.

Margin Interest Rate

The `InteractiveBrokersBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `InteractiveBrokersBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME
FutureOption	CME
Index	USA
IndexOption	USA
Forex	Oanda

Financial Advisors

IB supports FA accounts for Trading Firm and Institution organizations. FA accounts enable certified professionals to use a single trading algorithm to manage several client accounts.

To place trades using a subset of client accounts, create Account Groups in Trader Workstation and then define the `InteractiveBrokersOrderProperties` when you create orders.

```
self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
self.DefaultOrderProperties.FaGroup = "TestGroupEQ"
self.DefaultOrderProperties.FaMethod = "EqualQuantity"
self.DefaultOrderProperties.FaProfile = "TestProfileP"
self.DefaultOrderProperties.Account = "DU123456"
```

SecurityHolding objects aggregate your positions across all the account groups. If you have two groups where group A has 10 shares of SPY and group B has -10 shares of SPY, then `self.Portfolio["SPY"].Quantity` is zero.

The following table shows the supported allocation methods for FA group orders:

FaMethod	Description
"EqualQuantity"	Distributes shares equally between all accounts in the group. If you use this method, specify an order quantity.
"NetLiq"	Distributes shares based on the net liquidation value of each account. The system calculates ratios based on the net liquidation value in each account and allocates shares based on these ratios. If you use this method, specify an order quantity.
"AvailableEquity"	Distributes shares based on the amount of available equity in each account. The system calculates ratios based on the available equity in each account and allocates shares based on these ratios. If you use this method, specify an order quantity.
"PctChange"	Increases or decreases an already existing position. Positive percents increase positions and negative percents decrease positions. If you use this method, specify a percent instead of an order quantity.

PY

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
    self.DefaultOrderProperties.FaGroup = "TestGroupEQ"
    self.DefaultOrderProperties.FaMethod = "EqualQuantity"
    self.DefaultOrderProperties.FaProfile = "TestProfileP"
    self.DefaultOrderProperties.Account = "DU123456"

def OnData(self, slice: Slice) -> None:
    # Override the default order properties
    # "NetLiq" requires a order size input
    order_properties = InteractiveBrokersOrderProperties()
    order_properties.FaMethod = "NetLiq"
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    # "AvailableEquity" requires a order size input
    order_properties.FaMethod = "AvailableEquity"
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    # "PctChange" requires a percentage of portfolio input
    order_properties.FaMethod = "PctChange"
    self.SetHoldings(self.symbol, pct_portfolio, orderProperties=order_properties)
```

Supported Models

Kraken

Introduction

This page explains `KrakenBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.Kraken, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `KrakenBrokerageModel` supports trading Crypto .

Orders

The `KrakenBrokerageModel` supports several order types and order properties, but it doesn't support order updates.

Order Types

The following table describes the available order types for each asset class that the `KrakenBrokerageModel` supports:

Order Type	Crypto
MarketOrder	✓
LimitOrder	✓
LimitIfTouchedOrder	✓
StopMarketOrder	✓
StopLimitOrder	✓

Order Properties

The `KrakenBrokerageModel` supports custom order properties. The following table describes the members of the `KrakenOrderProperties` object that you can set to customize order execution:

Property	Description
`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`
`PostOnly`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.
`FeeInBase`	A flag to signal that the order fees should be paid in the base currency, which is the default behavior when selling. This flag must be the opposite of the `FeeInQuote` flag.
`FeeInQuote`	A flag to signal that the order fees should be paid in the quote currency, which is the default behavior when buying. This flag must be the opposite of the `FeeInBase` flag.
`NoMarketPriceProtection`	A flag to signal that no Market Price Protection should be used.
`ConditionalOrder`	An `Order` that's submitted when the primary order is executed. The `ConditionalOrder` quantity must match the primary order quantity and the `ConditionalOrder` direction must be the opposite of the primary order direction. This order property is only available for live algorithms.

PY

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = KrakenOrderProperties()
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled
    self.DefaultOrderProperties.PostOnly = False
    self.DefaultOrderProperties.FeeInBase = True
    self.DefaultOrderProperties.FeeInQuote = False
    self.DefaultOrderProperties.NoMarketPriceProtection = True

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = KrakenOrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    order_properties.PostOnly = True
    order_properties.FeeInBase = False
    order_properties.FeeInQuote = True
    order_properties.NoMarketPriceProtection = True
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(year, month, day))
    order_properties.PostOnly = False
    order_properties.FeeInBase = True
    order_properties.FeeInQuote = False
    order_properties.NoMarketPriceProtection = False
    order_properties.ConditionalOrder = StopLimitOrder(self.symbol, -quantity, stop_limit_price,
stop_price)
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `KrakenBrokerageModel` doesn't support order updates, but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages

That Don't Support Updates .

Fills

The `KrakenBrokerageModel` uses the ImmediateFillModel .

Slippage

The `KrakenBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
PY
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `KrakenBrokerageModel` uses the KrakenFeeModel .

Buying Power

The `KrakenBrokerageModel` uses the `CashBuyingPowerModel` for cash accounts and the `SecurityMarginModel` for margin accounts.

If you have a margin account, the `KrakenBrokerageModel` allows 1x leverage for most Crypto pairs. The following table shows pairs that have additional leverage available:

Quote Currency	Base Currencies	Leverage
ADA	BTC, ETH, USD, EUR	3
BCH	BTC, USD, EUR	2
BTC	USD, EUR	5
DASH	BTC, USD, EUR	3
EOS	BTC, ETH, USD, EUR	3
ETH	BTC, USD, EUR	5
LINK	BTC, ETH, USD, EUR	3
LTC	BTC, USD, EUR	3
REP	BTC, ETH, USD, EUR	2
TRX	BTC, ETH, USD, EUR	3
USDC	USD, EUR	3
USDT	USD, EUR	2
XMR	BTC, USD, EUR	2
XRP	BTC, USD, EUR	3
XTZ	BTC, ETH, USD, EUR	2

Settlement

The `KrakenBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `KrakenBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `KrakenBrokerageModel` is `Market.Kraken` .

Supported Models

Oanda

Introduction

This page explains `OandaBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.OandaBrokerage, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `OandaBrokerageModel` supports trading Forex and CFDs .

Orders

The `OandaBrokerageModel` supports several order types, the `TimeInForce` order property, and order updates.

Order Types

The following table describes the available order types for each asset class that the `OandaBrokerageModel` supports:

Order Type	Forex	CFD
MarketOrder	✓	✓
LimitOrder	✓	✓
StopMarketOrder	✓	✓

Time In Force

The `OandaBrokerageModel` the `GoodTilCanceled` TimeInForce .

```
self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled
self.LimitOrder(self.symbol, quantity, limit_price)
```

Updates

The `OandaBrokerageModel` supports order updates .

Fills

The `OandaBrokerageModel` uses the ImmediateFillModel .

Slippage

The `OandaBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
security.SetSlippageModel(ConstantSlippageModel(0))
```
PY

Fees

The `OandaBrokerageModel` uses the ConstantFeeModel with zero fees. OANDA offers "Spread" and "Core + Commission" pricing models. Our data is the spread dataset, so we model the spread pricing. To estimate OANDA Core+Commission data, use a custom fee model and a custom fill model .

```
security.SetFeeModel(ConstantFeeModel(0))
```
PY

Buying Power

The `OandaBrokerageModel` uses the `SecurityMarginModel` and allows up to 50x leverage.

Settlement

The `OandaBrokerageModel` uses the ImmediateSettlementModel for Forex trades and the AccountCurrencyImmediateSettlementModel for CFD trades.

Margin Interest Rate

The `OandaBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `OandaBrokerageModel` is `Market.Oanda` .

Supported Models

Samco

Introduction

This page explains `SamcoBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.Samco, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Samco, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `SamcoBrokerageModel` supports trading the following asset classes:

- Indian Equities
- India Equity Options
- India Futures

Orders

The `SamcoBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `SamcoBrokerageModel` supports:

Order Type	India Equity
MarketOrder	✓
LimitOrder	✓
StopMarketOrder	✓

Order Properties

The `SamcoBrokerageModel` supports custom order properties. The following table describes the members of the `IndiaOrderProperties` object that you can set to customize order execution:

Property	Description
`Exchange`	Select the exchange for sending the order to. The following instructions are available: • `NSE` • `BSE`
`ProductType`	A `ProductType` instruction to apply to the order. The `IndiaProductType` enumeration has the following members:
`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are available: • `Day` • `GoodTilCanceled` • `GoodTilDate`

PY

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = IndiaOrderProperties(Exchange.NSE,
IndiaOrderProperties.IndiaProductType.NRML)
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = IndiaOrderProperties(Exchange.BSE, IndiaOrderProperties.IndiaProductType.MIS)
    order_properties.TimeInForce = TimeInForce.Day
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties = IndiaOrderProperties(Exchange.BSE, IndiaOrderProperties.IndiaProductType.CNC)
    order_properties.TimeInForce = TimeInForce.GoodTilDate
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `SamcoBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The `SamcoBrokerageModel` uses the EquityFillModel .

Slippage

The `SamcoBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
PY
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `SamcoBrokerageModel` uses the SamcoFeeModel .

Buying Power

The `SamcoBrokerageModel` uses the `SecurityMarginModel` . If you have a margin account, the `SamcoBrokerageModel` allows up to 5x leverage.

Settlement

The `SamcoBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
PY
# For cash accounts:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For margin accounts:
security.SettlementModel = ImmediateSettlementModel()
```

Margin Interest Rate

The `SamcoBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `SamcoBrokerageModel` is `Market.India` .

Supported Models

TD Ameritrade

Introduction

This page explains `TDAmeritradeBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```py
self.SetBrokerageModel(BrokerageName.TDAmeritrade, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.TDAmeritrade, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `TDAmeritradeBrokerageModel` supports trading US Equities .

Orders

The `TDAmeritradeBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `TDAmeritradeBrokerageModel` supports:

Order Type	Equity
MarketOrder	✓
LimitOrder	✓
StopMarketOrder	✓
StopLimitOrder	✓

Time In Force

The `TDAmeritradeBrokerageModel` supports the following TimeInForce instructions:

- `Day`
- `GoodTilCanceled`
- `GoodTilDate`

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = TDAmeritradeOrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(year, month, day))
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `TDAmeritradeBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The `TDAmeritradeBrokerageModel` uses the EquityFillModel .

Slippage

The `TDAmeritradeBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `TDAmeritradeBrokerageModel` uses the TDAmeritradeFeeModel .

Buying Power

The `TDAmeritradeBrokerageModel` uses the `SecurityMarginModel` . If you have a margin account, the `TDAmeritradeBrokerageModel` allows up to 2x leverage.

Settlement

The `TDAmeritradeBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
# For cash accounts:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For margin accounts:
security.SettlementModel = ImmediateSettlementModel()
```

Margin Interest Rate

The `TDAmeritradeBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `TDAmeritradeBrokerageModel` is `Market.USA` .

Supported Models

Tradier

Introduction

This page explains `TradierBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `TradierBrokerageModel` supports trading US Equities and Equity Options .

Orders

The `TradierBrokerageModel` supports several order types, order properties, and most order updates.

Order Types

The following table describes the available order types for each asset class that the `TradierBrokerageModel` supports:

Order Type	Equity	Equity Options
MarketOrder	✓	✓
LimitOrder	✓	✓
StopMarketOrder	✓	✓
StopLimitOrder	✓	✓

Time In Force

The `TradierBrokerageModel` supports the following TimeInForce instructions:

- `Day`
- `GoodTilCanceled` (not available for short selling)
- `GoodTilDate`

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = OrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties.TimeInForce = TimeInForce.GoodTilDate(datetime(year, month, day))
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

PY

Updates

The `TradierBrokerageModel` supports most order updates . To update the quantity of an order, cancel the order and then submit a new order with the desired quantity. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Extended Market Hours

The `TradierBrokerageModel` doesn't support extended market hours trading. If you place an order outside of regular trading hours, the order will be processed at market open.

Automatic Cancellations

If you have open orders for a security when it performs a reverse split, the `TradierBrokerageModel` automatically cancels your orders.

Errors

To view the order-related error codes from Tradier, see Error Responses in their documentation.

The `TradierBrokerageModel` validates your orders for the following errors before sending them to Tradier:

Error	Description
`ShortOrderIsGtc`	You can't short sell with the `GoodTilCanceled` time in force
`SellShortOrderLastPriceBelow5`	You can't short sell stocks below $5
`IncorrectOrderQuantity`	The order quantity must be between 1 and 10,000,000 shares

Fills

The `TradierBrokerageModel` uses the EquityFillModel for Equity trades and the ImmediateFillModel for Option trades.

Slippage

The `TradierBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `TradierBrokerageModel` uses the ConstantFeeModel with zero fees.

```
security.SetFeeModel(ConstantFeeModel(0))
```

Buying Power

The `TradierBrokerageModel` uses the `OptionMarginModel` for Option trades and the `SecurityMarginModel` for Equity trades.

If you have a margin account, the `TradierBrokerageModel` allows 2x leverage for Equities.

Settlement

The `TradierBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
# For US Equities with a cash account:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For Equity Options with a cash account:
security.SettlementModel = DelayedSettlementModel(Option.DefaultSettlementDays,
Option.DefaultSettlementTime)

# For remaining cases:
security.SettlementModel = ImmediateSettlementModel()
```

Margin Interest Rate

The `TradierBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `TradierBrokerageModel` is `Market.USA` .

Supported Models

Trading Technologies

Introduction

This page explains `TradingTechnologiesBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```py
self.SetBrokerageModel(BrokerageName.TradingTechnologies, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.TradingTechnologies, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `TradingTechnologiesBrokerageModel` supports trading Futures .

Orders

The `TradingTechnologiesBrokerageModel` supports several order types, some `TimeInForce` order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `TradingTechnologiesBrokerageModel` supports:

Order Type	Futures
MarketOrder	✓
LimitOrder	✓
StopMarketOrder	✓
StopLimitOrder	✓

The `TradingTechnologiesBrokerageModel` enforces the following order rules:

- If you are buying (selling) with a `StopMarketOrder` or a `StopLimitOrder` , the stop price of the order must be greater (less) than the current security price.
- If you are buying (selling) with a `StopLimitOrder` , the limit price of the order must be greater (less) than the stop price.

Time In Force

The `TradingTechnologiesBrokerageModel` supports the `Day` and `GoodTilCanceled` TimeInForce order properties.

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = OrderProperties()
    order_properties.TimeInForce = TimeInForce.Day
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `TradingTechnologiesBrokerageModel` supports order updates .

Fills

The `TradingTechnologiesBrokerageModel` uses the FutureFillModel .

Slippage

The `TradingTechnologiesBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `TradingTechnologiesBrokerageModel` uses the ConstantFeeModel with zero fees.

```
security.SetFeeModel(ConstantFeeModel(0))
```

Buying Power

The `TradingTechnologiesBrokerageModel` uses the `FutureMarginModel` .

Settlement

The `TradingTechnologiesBrokerageModel` uses the FutureSettlementModel .

Margin Interest Rate

The `TradingTechnologiesBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `TradierBrokerageModel` is `Market.CME` .

Supported Models

Wolverine

Introduction

This page explains `WolverineBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.Wolverine, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Wolverine, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `WolverineBrokerageModel` supports trading US Equities .

Orders

The `WolverineBrokerageModel` supports one order type, but doesn't support order updates or extended market hours trading.

Order Types

The `WolverineBrokerageModel` supports market orders .

Updates

The `WolverineBrokerageModel` doesn't support order updates.

Extended Market Hours

The `WolverineBrokerageModel` doesn't support extended market hours trading. If you place an order outside of regular trading hours, the order is invalid.

Fills

The `WolverineBrokerageModel` uses the EquityFillModel for Equity trades.

Slippage

The `WolverineBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `WolverineBrokerageModel` uses the WolverineFeeModel .

Buying Power

The `WolverineBrokerageModel` uses the `SecurityMarginModel` . If you have a margin account, the `WolverineBrokerageModel` allows up to 2x leverage.

Settlement

The `WolverineBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```py
# For cash accounts:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For margin accounts:
security.SettlementModel = ImmediateSettlementModel()
```

Margin Interest Rate

The `WolverineBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `WolverineBrokerageModel` is `Market.USA` .

Supported Models

Zerodha

Introduction

This page explains `ZerodhaBrokerageModel` , including the asset classes it supports, its default security-level models , and it's default markets.

```
self.SetBrokerageModel(BrokerageName.Zerodha, AccountType.Cash)
self.SetBrokerageModel(BrokerageName.Zerodha, AccountType.Margin)
```

To view the implementation of this model, see the LEAN GitHub repository .

Asset Classes

The `ZerodhaBrokerageModel` supports trading Indian Equities .

Orders

We model the Zerodha API by supporting several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `ZerodhaBrokerageModel` supports:

Order Type	India Equity
MarketOrder	✓
LimitOrder	✓
StopMarketOrder	✓
StopLimitOrder	✓

Order Properties

The `ZerodhaBrokerageModel` supports custom order properties. The following table describes the members of the `IndiaOrderProperties` object that you can set to customize order execution:

Property	Description
`Exchange`	Select the exchange for sending the order to. The following instructions are supported: • `NSE` • `BSE`
`ProductType`	A `ProductType` instruction to apply to the order. The `IndiaProductType` enumeration has the following members:
`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`

```py
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = IndiaOrderProperties(Exchange.NSE,
IndiaOrderProperties.IndiaProductType.NRML)
    self.DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled

def OnData(self, slice: Slice) -> None:
    # Use default order order properties
    self.LimitOrder(self.symbol, quantity, limit_price)

    # Override the default order properties
    order_properties = IndiaOrderProperties(Exchange.BSE, IndiaOrderProperties.IndiaProductType.MIS)
    order_properties.TimeInForce = TimeInForce.Day
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    order_properties = IndiaOrderProperties(Exchange.BSE, IndiaOrderProperties.IndiaProductType.CNC)
    order_properties.TimeInForce = TimeInForce.GoodTilDate
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)
```

Updates

The `ZerodhaBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The `ZerodhaBrokerageModel` uses the EquityFillModel .

Slippage

The `ZerodhaBrokerageModel` uses the ConstantSlippageModel with zero slippage.

```
PY
security.SetSlippageModel(ConstantSlippageModel(0))
```

Fees

The `ZerodhaBrokerageModel` uses the ZerodhaFeeModel .

Buying Power

The `ZerodhaBrokerageModel` uses the `SecurityMarginModel` . If you have a margin account, the `ZerodhaBrokerageModel` allows up to 5x leverage.

Settlement

The `ZerodhaBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
PY
# For cash accounts:
security.SettlementModel = DelayedSettlementModel(Equity.DefaultSettlementDays,
Equity.DefaultSettlementTime)

# For margin accounts:
security.SettlementModel = ImmediateSettlementModel()
```

Margin Interest Rate

The `ZerodhaBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `ZerodhaBrokerageModel` is `Market.India` .

Reality Modeling

Buying Power

Introduction

Buying power models (also known as margin models) control how much buying power or leverage your portfolio has to make trades. When you place an order, LEAN uses the buying power model to determine whether the order should be submitted so you avoid placing orders that would be rejected by the brokerage. Buying power calculations can be very complex and depend on many factors, including the brokerage or even the time of day. For example, the `PatternDayTradingMarginModel` lets you have 4x leverage during regular trading hours and 2x leverage during pre-market and post-market trading hours.

What is Buying Power?

Buying power is the total amount of money you can spend in your brokerage account. It depends on the security type and account type. On one hand, Option and Future contracts are leveraged securities with specific buying power rules. On the other hand, the buying power of cash accounts is just the cash in the account while the buying power of margin accounts is leveraged by the brokerage credit.

What is Margin?

Margin is a credit loan from your brokerage you receive after you deposit collateral into your margin account. You need margin to place short-biased trades and you can use margin to increase your buying power, but you incur interest fees. **Margin** is the dollar value of the loan that the brokerage gives you. A **margin requirement** of 25% means that to purchase 10,000 worth of securities, you need at least 2,500 worth of collateral in your brokerage account to open the trade and you can borrow the rest on margin. **Maintenance margin** is the minimum equity (equity = total portfolio value - borrowed funds) you must have in your brokerage account to stay in your positions. If the value of your portfolio falls below the maintenance margin, you receive a margin call . If you receive a margin call, you either need to add more capital to your brokerage account or the brokerage will liquidate some of your holdings to reduce your exposure and their risk.

Some securities have special margin rules. Derivatives are leveraged assets with a floating margin requirement. In particular, long Options have zero maintenance margin requirement and their initial margin requirement is only the premium that you pay upfront.

What is Leverage?

Leverage is using borrowed money to increase your buying power. Leverage has an inverse relationship with your margin requirement and maintenance margin. If you have a margin requirement of 50%, you can use up to 1 / 50% = 2 leverage. Trading with leverage can be risky. It can boost your returns on profitable trades but can make your losing trades more expensive. If you have 10,000 in your brokerage margin account and purchase 20,000 worth of securities, you are trading with 2x leverage. If the value of the securities in your portfolio drops by 50% when you have a 2x leverage position, you lose all of your equity.

What Are Position Groups?

A position group is the combination of holdings. It has lower margin requirement and maintenance margin than the sum of each position of the group. If you have a margin requirement of 29,150 to purchase an in-the-money call Option contract, and a margin requirement of 101,499 to sell an out-of-the money call Option contract, the total margin requirement is $130,649. However, these positions compose a bull call spread with a margin requirement of $0.

Set Models

The brokerage model of your algorithm automatically sets the buying power model for each security, but you can override it. To manually set the buying power model of a security, call the `SetBuyingPowerModel` method on the `Security` object.

```
PY
# In Initialize
security = self.AddEquity("SPY")
security.SetFeeModel(SecurityMarginModel(3))
```

You can also set the buying power model in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetBuyingPowerModel(SecurityMarginModel(3))
```

You cannot change the position group buying power models.

Supported Models

The following buying power models are available:

- BuyingPowerModel
- CashBuyingPowerModel
- ConstantBuyingPowerModel
- CryptoFutureMarginModel
- NullBuyingPowerModel
- FutureMarginModel
- FuturesOptionsMarginModel

- SecurityMarginModel
- OptionMarginModel
- PatternDayTradingMarginModel

The following position group buying power models are available:

- SecurityPositionGroupBuyingPowerModel
- OptionStrategyPositionGroupBuyingPowerModel

Default Behavior

The brokerage model of your algorithm automatically sets the buying power model of each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	**Model**
Equity Options	`OptionMarginModel`
Futures	`FutureMarginModel`
Future Options	`FuturesOptionsMarginModel`
Index Options	`OptionMarginModel`
Crypto	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
CryptoFuture	`CryptoFutureMarginModel`
Forex	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
Other	`SecurityMarginModel`

Model Structure

Buying power models should extend the `BuyingPowerModel` class. Extensions of the `BuyingPowerModel` class should implement the following methods:

```py
class MyBuyingPowerModel(BuyingPowerModel):
    def __init__(self,
                 leverage = 2: float,
                 requiredFreeBuyingPowerPercent = 0: float):
        super().__init__(leverage, requiredFreeBuyingPowerPercent)

    def GetLeverage(self, security: Security) -> float:
        return super().GetLeverage(security)

    def SetLeverage(self, security: Security, leverage: float) -> None:
        super().SetLeverage(security, leverage)

    def GetInitialMarginRequiredForOrder(self,
                 parameters: InitialMarginRequiredForOrderParameters) -> InitialMargin:
        return super().GetInitialMarginRequiredForOrder(parameters)

    def GetMaintenanceMargin(self,
                 parameters: MaintenanceMarginParameters) -> MaintenanceMargin:
        return super().GetMaintenanceMargin(parameters)

    def GetMarginRemaining(self,
                 portfolio: SecurityPortfolioManager,
                 security: Security,
                 direction: OrderDirection) -> float:
        return super().GetMarginRemaining(portfolio, security, direction)

    def GetInitialMarginRequirement(self,
                 parameters: InitialMarginParameters) -> InitialMargin:
        return super().GetInitialMarginRequirement(parameters)

    def HasSufficientBuyingPowerForOrder(self,
                 parameters: HasSufficientBuyingPowerForOrderParameters
        ) -> HasSufficientBuyingPowerForOrderResult:
        return super().HasSufficientBuyingPowerForOrder(parameters)

    def GetMaximumOrderQuantityForDeltaBuyingPower(self,
                 parameters: GetMaximumOrderQuantityForDeltaBuyingPowerParameters
        ) -> GetMaximumOrderQuantityResult:
        return super().GetMaximumOrderQuantityForDeltaBuyingPower(parameters)

    def GetMaximumOrderQuantityForTargetBuyingPower(self,
                 parameters: GetMaximumOrderQuantityForTargetBuyingPowerParameters
        ) -> GetMaximumOrderQuantityResult:
        return super().GetMaximumOrderQuantityForTargetBuyingPower(parameters)

    def GetReservedBuyingPowerForPosition(self,
                 parameters: ReservedBuyingPowerForPositionParameters
        ) -> ReservedBuyingPowerForPosition:
        return super().GetReservedBuyingPowerForPosition(parameters)

    def GetBuyingPower(self,
                 parameters: BuyingPowerParameters) -> BuyingPower:
        return super().GetBuyingPower(parameters)
```

Disable Buying Power

You can disable order margin checks and opt to let your brokerage decide to accept or reject the trades. This is helpful in live trading if you have a more permissive brokerage margin allowance that what LEAN models. The default position group buying power models are helpful for Option trading strategies. However, it can be counterproductive if it's not a supported Option strategy . To disable the validations of the default position group buying power model, use the `NullSecurityPositionGroupModel` . To set the `NullSecurityPositionGroupModel` for the portfolio, during initialization , call the `SetPositions` method with the `SecurityPositionGroupModel.Null` argument.

```py
self.Portfolio.SetPositions(SecurityPositionGroupModel.Null)
```

To disable the validations of the default buying power model , use the `NullBuyingPowerModel` . To set the `NullBuyingPowerModel` for a security subscription, call the `SetBuyingPowerModel` method with the `BuyingPowerModel.Null` argument.

```
PY
equity = self.AddEquity("SPY")
equity.SetBuyingPowerModel(BuyingPowerModel.Null)
# Alias:
# equity.SetMarginModel(SecurityMarginModel.Null)
```

You can also set the asset `NullBuyingPowerModel` in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to set the buying power of securities in the security initializer, call `SetSecurityInitializer` before you create security subscriptions and after you call `SetBrokerageModel` .

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the security buying power
        security.SetBuyingPowerModel(BuyingPowerModel.Null)
```

Set Asset Leverage

The buying power model sets the leverage for each security in your algorithm, but you can override its leverage settings after the buying power model is set.

To set the leverage when you create a security subscription, pass in a `leverage` argument.

```
PY
# In Initialize
AddEquity("SPY", leverage=3)
```

You can also set the asset leverage in a security initializer. In order to set the leverage of securities in the security initializer, call `SetSecurityInitializer` before you create security subscriptions and before you call `SetBrokerageModel` . If you pass in a `leverage` argument when you create the security subscription, the `leverage` argument takes precedence over the `SetLeverage` call in the security initializer.

```
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the security leverage
        security.SetLeverage(3)
```

To set the leverage for all securities in a universe, set the `UniverseSettings.Leverage` property.

```
# In Initialize
self.UniverseSettings.Leverage = 3
```

In live trading, LEAN doesn't ignore the leverage you set. However, if you set a different leverage from what your brokerage provides, it creates a mismatch between the buying power in your algorithm and the buying power the brokerage gives you. In this case, orders can pass the validations in LEAN but your brokerage may reject them.

PDT Rule

If all of the following statements are true, you are classified as a pattern day trader:

- You reside in the United States.
- You trade in a margin account.
- You execute 4+ intraday US Equity trades within 5 business days.
- Your intraday US Equity trades represent more than 6% of your total trades.

Pattern day traders must maintain a minimum equity of $25,000 in their margin account to continue trading. For more information about pattern day trading, see Am I a Pattern Day Trader? on the FINRA website.

The `PatternDayTradingMarginModel` doesn't enforce minimum equity rules and doesn't limit your trades, but it adjusts your available leverage based on the market state. During regular market hours, you can use up to 4x leverage. During extended market hours, you can use up to 2x leverage.

```
security.MarginModel = PatternDayTradingMarginModel()
```

Examples

Demonstration Algorithms
CustomBuyingPowerModelAlgorithm.py Python CustomModelsAlgorithm.py Python
NullBuyingPowerOptionBullCallSpreadAlgorithm.py Python

Reality Modeling

Settlement

Settlement

Key Concepts

Introduction

After you trade an asset, the brokerage needs to settle the funds in your account. The most common type of settlement is immediate, where the funds are immediately available for trading after the transaction. In some cases, you may have delayed settlement, where you sell an asset and need to wait a few days to spend the cash you receive from the sale. A settlement model simulates these settlement rules.

Set Models

The brokerage model of your algorithm automatically sets the settlement model for each security, but you can override it. To manually set the settlement model of a security, set the `SettlementModel` property on the `Security` object.

```
PY
# In Initialize
security = self.AddEquity("SPY")
# Set a delayed settlement model that settles 7 days after the trade at 8 AM
security.SettlementModel = DelayedSettlementModel(7, timedelta(hours=8))
```

You can also set the settlement model in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SettlementModel = DelayedSettlementModel(7, timedelta(hours=8))
```

To view all the pre-built settlement models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the settlement model for each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the settlement model based on the security type and your account type. The following table shows how it sets the settlement models:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `DefaultBrokerageModel` uses the ImmediateSettlementModel .

The default delayed settlement rule for US Equity trades is T+2 at 8 AM Eastern Time (ET). For example, if you sell on Monday, the trade settles on Wednesday at 8 AM. The default delayed settlement rule for Future and Option contracts is T+1 at 8 AM.

Model Structure

Settlement models must extend the `ISettlementModel` interface. Extensions of the `ISettlementModel` interface must implement the `Scan` and `ApplyFunds` methods. The `Scan` method is automatically called at the top of each hour and it receives a `ScanSettlementModelParameters` object. The `ApplyFunds` method receives an `ApplyFundsSettlementModelParameters` object and applies the settlement rules. The `ApplyFunds` method is also automatically called, but its frequency depends on the security type.

The `ApplyFunds` method is automatically called when you fill an order for the following security types:

- Equity
- Equity Options
- Crypto
- Forex
- Future Options
- Index Options

The `ApplyFunds` method is automatically called when you close a position for the following security types:

- Futures
- Crypto Futures
- CFD

`ApplyFundsSettlementModelParameters` objects have the following properties:

`ScanSettlementModelParameters` objects have the following properties:

You likely don't need to create a custom settlement model because the supported models already implement immediate and delayed settlement rules.

Settlement

Supported Models

Introduction

This page describes all of the pre-built settlement models in LEAN. For more brokerage-specific settlement models, see the brokerage model documentation . If none of these models perform exactly how you want, create a custom settlement model .

Immediate Model

The `ImmediateSettlementModel` immediately adds or removes the cash from your portfolio when your transactions fill.

```
security.SettlementModel = ImmediateSettlementModel()
```

To view the implementation of this model, see the LEAN GitHub repository .

Delayed Model

The `DelayedSettlementModel` immediately removes the cash from your portfolio when your buy orders fill. When your sell orders fill, it adds the cash to your unsettled cash book . When the settlement period ends, the unsettled cash is added to your portfolio.

```
security.SettlementModel = DelayedSettlementModel(7, timedelta(hours=8))
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`numberOfDays`	`int`	The number of days required for settlement	
`timeOfDay`	`timedelta`	The time of day used for settlement	

To view the implementation of this model, see the LEAN GitHub repository .

Account Currency Immediate Model

The `AccountCurrencyImmediateSettlementModel` applies cash settlement immediately and automatically converts the settlement cash into the account currency.

```
security.SettlementModel = AccountCurrencyImmediateSettlementModel()
```

To view the implementation of this model, see the LEAN GitHub repository .

Future Model

The `FutureSettlementModel` settles the daily profit and loss at the start of each day.

```
security.SettlementModel = FutureSettlementModel()
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Reality Modeling

Options Models

Options Models

Pricing

Introduction

Option price models compute the theoretical price of Option contracts, their implied volatility, and their Greek values. Theoretical prices can help you detect undervalued and overvalued contracts, implied volatility can provide you insight into the upcoming volatility of the underlying security, and Greek values can help you hedge your portfolio.

What are Greeks?

Option Greeks measure the exposure of Option price or Option delta to the movement of different factors such as the underlying price, time, and volatility. The Greeks are a function of implied volatility. For more information about them, see The Greek Letters .

What Is Implied Volatility?

Implied volatility is the forecasted future volatility of a security. The Option price model uses the realized volatility to calculate the implied volatility. By default, it uses the formula from Brenner and Subrahmanyam (1988) as the initial guess for implied volatility.

$$\frac{P}{S} \sqrt{\frac{2\pi}{T}}$$

where P is the Option contract price, S is the underlying price, and T is the time until Option expiration.

If the volatility model of the underlying security is ready , the price model uses its value as the initial guess for implied volatility and as an input to calculate the theoretical contract prices.

Set Models

To set the pricing model of an Option, set its `PriceModel` property.

If you have access to the `Option` object when you subscribe to the Option universe or contract, you can set the price model immediately after you create the subscription.

```py
# In Initialize
option = self.AddOption("SPY")
option.PriceModel = OptionPriceModels.CrankNicolsonFD()
```

Otherwise, set the price model in a security initializer .

```py
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the price model
        if security.Type == SecurityType.Option: # Option type
            security.PriceModel = OptionPriceModels.CrankNicolsonFD()
```

Supported Models

LEAN supports the following Option price models. QLNet provides the underlying implementation of these models.

Model	American Style	European Style
`AdditiveEquiprobabilities`	✓	✓
`BaroneAdesiWhaley`	✓	
`BinomialCoxRossRubinstein`	✓	✓
`BinomialJarrowRudd`	✓	✓
`BinomialJoshi`	✓	✓
`BinomialLeisenReimer`	✓	✓
`BinomialTian`	✓	✓
`BinomialTrigeorgis`	✓	✓
`BjerksundStensland`	✓	
`BlackScholes`		✓
`CrankNicolsonFD`	✓	✓
`Integral`		✓

If you set the price model of an Option to a model with an incompatible style, LEAN throws an exception.

Default Behavior

The default Option pricing model is the `BjerksundStensland` for American Options or `BlackScholes` for European Options.

Disable Pricing

To turn off the Option price model, use the `CurrentPriceOptionPriceModel` . This model sets the Greeks to 0, sets the implied volatility to 0, and sets the theoretical price to the current price.

```
option.PriceModel = CurrentPriceOptionPriceModel()
```

PY

Examples

Demonstration Algorithms
BasicTemplateOptionsHistoryAlgorithm.py Python BasicTemplateOptionsPriceModel.py Python

Options Models

Volatility

Volatility

Key Concepts

Introduction

Volatility models measure the historical volatility of an asset. They are mostly used to calculate the volatility of the underlying security of an Option because the implied volatility of an Option contract needs an initial guess. The historical volatility doesn't need to be the standard deviation of the asset prices. The various volatility models in LEAN each have a unique methodology to calculate volatility.

Set Models

To set the volatility model of the underlying security of an Option, set the `VolatilityModel` property of the `Security` object. The volatility model can have a different resolution than the underlying asset subscription.

```
PY
# In Initialize
underlying_security = self.AddEquity("SPY")
underlying_security.VolatilityModel = StandardDeviationOfReturnsVolatilityModel(30)
```

You can also set the volatility model in a security initializer . If your algorithm has a universe of underlying assets, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the volatility model
        if security.Type == SecurityType.Equity:
            security.VolatilityModel = StandardDeviationOfReturnsVolatilityModel(30)
```

To view all the pre-built volatility models, see Supported Models .

Default Behavior

The default underlying volatility model for Equity Options and Index Options is the StandardDeviationOfReturnsVolatilityModel based on 30 days of daily resolution data. The default underlying volatility model for Future Options is the NullVolatilityModel .

Model Structure

Volatility models should extend the `BaseVolatilityModel` class. Extensions of the `BaseVolatilityModel` class must have `Update` and `GetHistoryRequirements` methods and a `Volatility` property. The `Update` method receives `Security` and `BaseData` objects and then updates the `Volatility` . The `GetHistoryRequirements` method receives `Security` and `datetime` objects and then returns a list of `HistoryRequest` objects that represent the history requests to warm up the model. Volatility models receive data at each time step in the algorithm to update their state.

```
PY
class MyVolatilityModel(BaseVolatilityModel):
    Volatility: float = 0

    def SetSubscriptionDataConfigProvider(self,
         subscriptionDataConfigProvider: ISubscriptionDataConfigProvider) -> None:
        SubscriptionDataConfigProvider = subscriptionDataConfigProvider

    def Update(self, security: Security, data: BaseData) -> None:
        pass

    def GetHistoryRequirements(self,
         security: Security,
         utcTime: datetime,
         resolution: Resolution = None,
         barCount: int = None) -> List[HistoryRequest]:
        return super().GetHistoryRequirements(security, utcTime, resolution, barCount)
```

Warm Up Models

To use your volatility model as the inital guess for the implied volatility , warm up the volatility model of the underlying security. If you subscribe to all the Options in the `Initialize` method, set a warm-up period to warm up their volatility models. The warm-up period should provide the volatility models with enough data to compute their values.

```
PY
# In Initialize
self.SetWarmUp(30, Resolution.Daily)

# In OnData
if self.IsWarmingUp:
    return
```

If you have a dynamic universe of underlying assets and add Option contracts to your algorithm with the `AddOptionContract` , `AddIndexOptionContract` , or `AddFutureOptionContract` methods, warm up the volatility model when the underlying asset enters your universe. We recommend you do this inside a security initializer .

```py
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices), self))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder, algorithm:
QCAlgorithm) -> None:
        super().__init__(brokerage_model, security_seeder)
        self.algorithm = algorithm

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite and warm up the volatility model
        if security.Type == SecurityType.Equity:  # Underlying asset type
            security.VolatilityModel = StandardDeviationOfReturnsVolatilityModel(30)
            trade_bars = self.algorithm.History[TradeBar](security.Symbol, 30, Resolution.Daily)
            for trade_bar in trade_bars:
                security.VolatilityModel.Update(security, trade_bar)
```

Examples

Demonstration Algorithms
CustomVolatilityModelAlgorithm.py Python

Volatility

Supported Models

Introduction

This page describes all of the pre-built volatility models in LEAN. If none of these models perform exactly how you want, create a custom volatility model .

Null Model

The `NullVolatilityModel` sets the volatility of the security to zero. It's the default volatility model for the underlying asset of Future Options.

```
PY
underlying_security.VolatilityModel = VolatilityModel.Null
```

To view the implementation of this model, see the LEAN GitHub repository .

Standard Deviation of Returns Model

The `StandardDeviationOfReturnsVolatilityModel` sets the volatility of the security to the annualized sample standard deviation of trailing returns. It's the default volatility model for the underlying asset of Equity Options and Index Options.

```
PY
underlying_security.VolatilityModel = StandardDeviationOfReturnsVolatilityModel(30)
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`periods`	`int`	The max number of samples to use when calculating the standard deviation of returns. This value must be greater than two.	
`resolution`	`Resolution` / `NoneType`	The resolution of the price data used to calculate the standard deviation. This only has a material effect in live mode. For backtesting, this value does not cause any behavioral changes.	`None`
`updateFrequency`	`timedelta` / `NoneType`	The frequency at which new values are inserted into the rolling window for the standard deviation calculation. If the value is `None` , it defaults to the `timedelta` representation of `resolution` . If the value and `resolution` are `None` , it defaults to a `timedelta` of one day.	`None`

To view the implementation of this model, see the LEAN GitHub repository .

Relative Standard Deviation Model

The `RelativeStandardDeviationVolatilityModel` sets the volatility of the security to the relative standard deviation of its price. In symbols, the value is

$$\frac{|\mu|}{\sigma}$$

where μ is the average of the samples and σ is the standard deviation of the samples.

```
underlying_security.VolatilityModel = RelativeStandardDeviationVolatilityModel(timedelta(days=1), 10)
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`periodSpan`	`timedelta`	The period of time to wait between each sample of the security price	
`periods`	`int`	The number of samples to use to calculate the volatility value	

To view the implementation of this model, see the LEAN GitHub repository .

Options Models

Exercise

Introduction

If you exercise a long Option position or are assigned on your short Option position, LEAN processes an Option exercise order. The Option exercise model converts the Option exercise order into an OrderEvent .

Set Models

To set the exercise model of an Option, call the `SetOptionExerciseModel` method of the `Option` object.

If you have access to the `Option` object when you subscribe to the Option universe or contract, you can set the exercise model immediately after you create the subscription.

```py
# In Initialize
option = AddOption("SPY")
option.SetOptionExerciseModel(DefaultExerciseModel())
```

Otherwise, set the assignment model in a security initializer .

```py
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the exercise model
        if security.Type == SecurityType.Option: # Option type
            option.SetOptionExerciseModel(DefaultExerciseModel())
```

Default Behavior

The default Option exercise model is the `DefaultExerciseModel` . The `DefaultExerciseModel` fills exercise orders to the full quantity with zero fees and applies an order tag to represent if the order is an exercise or assignment. To view the implementation of this model, see the LEAN GitHub repository .

Model Structure

Option exercise models must implement the `OptionExercise` method, which receives `Option` and `OptionExerciseOrder` objects and then returns a list of `OrderEvent` objects that contain the order fill information.

```py
class MyExerciseModel:
    def OptionExercise(self, option: Option, order: OptionExerciseOrder) -> List[OrderEvent]:
        in_the_money = option.IsAutoExercised(option.Underlying.Close)
        is_assignment = in_the_money and option.Holdings.IsShort

        order_event = OrderEvent(
            order.Id,
            option.Symbol,
            Extensions.ConvertToUtc(option.LocalTime, option.Exchange.TimeZone),
            OrderStatus.Filled,
            Extensions.GetOrderDirection(order.Quantity),
            0.0,
            order.Quantity,
            OrderFee.Zero,
            "Tag"
        )
        order_event.IsAssignment = is_assignment
        return [ order_event ]
```

`OptionExerciseOrder` objects have the following properties:

The following table describes the arguments of the `OrderEvent` constructor:

Argument Details
Argument: `orderId` Id of the parent order Data Type: `int` | Default Value: -
Argument: `symbol` Asset Symbol Data Type: `Symbol` | Default Value: -
Argument: `utcTime` Date/time of this event Data Type: `datetime` | Default Value: -
Argument: `direction` The direction of the order. The `OrderDirection` enumeration has the following members: Data Type: `OrderDirection` | Default Value: Hold

Argument Details
Argument: `fillPrice` Fill price information if applicable Data Type: `float` | Default Value: 0
Argument: `fillQuantity` Fill quantity Data Type: `float` | Default Value: 0
Argument: `orderFee` The order fee. You can use `OrderFee.Zero` or create an `OrderFee` object with a custom fee. `OrderFee(CashAmount(0.5, 'USD'))` Data Type: `OrderFee` | Default Value: -
Argument: `message` Message from the exchange Data Type: `str` | Default Value: ""

`OrderEvent` objects have the following attributes:

Examples

Demonstration Algorithms
CustomOptionExerciseModelRegressionAlgorithm.py Python

Options Models

Assignment

Introduction

If you sell an Option in a backtest, an assignment model can simulate an Option exercise order on behalf of the buyer and assign you to complete the requirements of the contract.

Set Models

To set the assignment model of an Option, call the `SetOptionAssignmentModel` method of the `Option` object.

If you have access to the `Option` object when you subscribe to the Option universe or contract, you can set the assignment model immediately after you create the subscription.

```
PY
# In Initialize
option = AddOption("SPY")
option.SetOptionAssignmentModel(DefaultOptionAssignmentModel())
```

Otherwise, set the assignment model in a security initializer .

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the assignment model
        if security.Type == SecurityType.Option: # Option type
            option.SetOptionAssignmentModel(DefaultOptionAssignmentModel())
```

Default Behavior

The default Option assignment model is the `DefaultOptionAssignmentModel` . The `DefaultOptionAssignmentModel` scans your portfolio every hour. It considers exercising American-style Options if they are within 4 days of their expiration and it considers exercising European-style Options on their day of expiration. If you have sold an Option that's 5% in-the-money and the Option exercise order is profitable after the cost of fees, this model exercises the Option.

To view the implementation of this model, see the LEAN GitHub repository .

Model Structure

Option assignment models should implement the `IOptionAssignmentModel` interface. Extensions of the `IOptionAssignmentModel` interface must implement the `GetAssignment` method, which automatically fires at the top of each hour and returns the Option assignments to generate.

```
class MyOptionAssignmentModel:

    def GetAssignment(self, parameters: OptionAssignmentParameters) -> OptionAssignmentResult:
        option = parameters.Option
        if self.IsInTheMoney(option):
            return OptionAssignmentResult(option.Holdings.AbsoluteQuantity, "tag")
        return OptionAssignmentResult.Null
```

PY

The `OptionAssignmentParameters` object has the following members:

To exercise the Option, return an `OptionAssignmentResult` with a positive quantity. Otherwise, return `OptionAssignmentResult.Null` . The `OptionAssignmentResult` constructor accepts the following arguments:

Argument	Data Type	Description	Default Value
`quantity`	`float`	The quantity to assign	
`tag`	`str`	The order tag to use	

Disable Assignments

To disable Option assignments, set the Option assignment model to the `NullOptionAssignmentModel` .

```
option.SetOptionAssignmentModel(NullOptionAssignmentModel())
```

PY

Examples

Demonstration Algorithms
CustomOptionAssignmentRegressionAlgorithm.py Python

Reality Modeling

Margin Interest Rate

Margin Interest Rate

Key Concepts

Introduction

Margin interest is a cost associated with trading on margin. Margin interest rate models model margin interest cash flows by directly adding or removing cash from your portfolio.

Set Models

The brokerage model of your algorithm automatically sets the margin interest rate model for each security, but you can override it. To manually set the margin interest rate model of a security, assign a model to the `MarginInterestRateModel` property of the Security object.

```
PY
# In Initialize
security = self.AddEquity("SPY")
security.SetMarginInterestRateModel(MarginInterestRateModel.Null)
```

You can also set the margin interest rate model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `SetSecurityInitializer` before you create the subscriptions.

```
PY
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetMarginInterestRateModel(MarginInterestRateModel.Null)
```

To view all the pre-built margin interest rate models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the margin interest rate model of each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the NullMarginInterestRateModel .

Model Structure

Margin interest rate models should implement the `IMarginInterestRateModel` interface. Extensions of the `IMarginInterestRateModel` interface must implement the `ApplyMarginInterestRate` method, which applies margin interest payments to the portfolio.

```
PY
class MyMarginInterestRateModel:

    def ApplyMarginInterestRate(self, marginInterestRateParameters: MarginInterestRateParameters) -> None:
        holdings = marginInterestRateParameters.Security.Holdings
        position_value = holdings.GetQuantityValue(holdings.Quantity)
        position_value.Cash.AddAmount(-1)
```

The `ApplyMarginInterestRate` method is automatically called at the top of each hour.

Margin Interest Rate

Supported Models

Introduction

This page describes the pre-built margin interest rate models in LEAN. If none of these models perform exactly how you want, create a custom margin interest rate model .

Null Model

The `NullMarginInterestRateModel` doesn't charge any margin interest. It's the default margin interest rate of the DefaultBrokerageModel .

```
PY
security.MarginInterestRateModel = MarginInterestRateModel.Null
```

To view the implementation of this model, see the LEAN GitHub repository .

Binance Futures Model

The `BinanceFutureMarginInterestRateModel` simulates the margin cost and payments of your Crypto Future holdings. When the funding rate is positive, the price of the perpetual contract is higher than the mark price, so traders who are long pay for short positions. Conversely, a negative funding rate indicates that perpetual prices are below the mark price, so traders who are short pay for long positions. The interest amount is

Nominal \ Value \ of \ Positions * Funding \ Rate

The interest amount is periodically credited or debited from your currency holdings while you hold open positions. The interest amount is charged at 12 AM, 8 AM, 4 PM in your algorithm time zone .

```
PY
security.MarginInterestRateModel = BinanceFutureMarginInterestRateModel()
```

To view the implementation of this model, see the LEAN GitHub repository .

Reality Modeling

Margin Calls

Introduction

If the value of your portfolio falls below the maintenance margin, you receive a margin call. If you receive a margin call, you either need to add more capital to your brokerage account or the brokerage will liquidate some of your holdings to reduce your exposure and their risk. A margin call model monitors the margin levels of your portfolio, issues margin call warnings, and submits orders when margin calls occur.

Set Models

To set the margin call model, set the `MarginCallModel` property of the `Portfolio` object.

```
PY
# In Initialize
self.Portfolio.MarginCallModel = DefaultMarginCallModel(self.Portfolio, self.DefaultOrderProperties)
```

Default Behavior

The brokerage model of your algorithm automatically sets the margin call model for the portfolio. The default brokerage model is the `DefaultBrokerageModel` , which sets the `DefaultMarginCallModel` . The `DefaultMarginCallModel` issues margin call warnings when the margin remaining in your portfolio is less than or equal to 5% of the total portfolio value. When a margin call occurs, this model sorts the generated margin call orders in ascending order by their unrealized profit and then executes each order synchronously until your portfolio is within the margin requirements.

Model Structure

Margin call models must extend the `DefaultMarginCallModel` class. Extensions of the `DefaultMarginCallModel` class can override the `GetMarginCallOrders` and `ExecuteMarginCall` methods.

The `GetMarginCallOrders` method scans the portfolio and the updated data for a potential margin call situation that may get the holdings below zero. The method must return a list of `SubmitOrderRequest` objects that represent the margin call orders. To issue a margin call warning during this method, set the `issueMarginCallWarning` argument of the method to true.

The `ExecuteMarginCall` method receives the list of `SubmitOrderRequest` objects from the `GetMarginCallOrders` method, executes some of them, and returns a list of `OrderTicket` objects.

```
class MyMarginCallModel(DefaultMarginCallModel):
    def __init__(self,
                 portfolio: SecurityPortfolioManager,
                 defaultOrderProperties: IOrderProperties):
        super().__init__(portfolio, defaultOrderProperties)

    def ExecuteMarginCall(self,
                 generatedMarginCallOrders: List[SubmitOrderRequest]) -> List[OrderTicket]:
        return super().ExecuteMarginCall(generatedMarginCallOrders)

    def GetMarginCallOrders(self,
                 issueMarginCallWarning: bool) -> List[SubmitOrderRequest]:
        return super().GetMarginCallOrders(issueMarginCallWarning)
```

PY

Disable Margin Calls

To disable margin calls, set the margin call model to the `NullMarginCallModel` .

```
self.Portfolio.MarginCallModel = MarginCallModel.Null
```

PY

Monitor Margin Call Events

When the margin call model of your portfolio issues a margin call warning, we notify your algorithm through the `OnMarginCallWarning` event handler.

```
def OnMarginCallWarning(self) -> None:
    self.Debug(f"Warning: Close to margin call")
```

PY

Before we send the orders that the margin call model produces, we notify your algorithm through the `OnMarginCall` event handler. This notification gives your algorithm a chance to liquidate some positions or modify the margin call orders. To modify the orders, adjust the list of `SubmitOrderRequest` objects the event handler receives.

```
def OnMarginCall(self, requests: List[SubmitOrderRequest]) -> List[SubmitOrderRequest]:
    for i, order in enumerate(requests):
        # liquidate an extra 10% each time you get a margin call to give yourself more padding
        new_quantity = int(order.Quantity * 1.1)
        requests[i] = SubmitOrderRequest(order.OrderType, order.SecurityType,
                                         order.Symbol, new_quantity, order.StopPrice,
                                         order.LimitPrice, 0, self.Time, "OnMarginCall")
    return requests
```

PY

Submit Order Request

If you receive a margin call or create a margin call model, you'll need to work with `SubmitOrderRequest` objects. These objects have the following properties:

The following table describes the arguments of the `SubmitOrderRequest` constructor:

Argument	Data Type	Description	Default Value
orderType	OrderType	The order type to be submitted	
securityType	SecurityType	The security type of the asset	
symbol	Symbol	The symbol to be traded	
quantity	float	The number of units to order	
stopPrice	float	The stop price for stop orders	
limitPrice	float	The limit price for limit orders	
triggerPrice	float	The trigger price for limit if touched orders	
time	datetime	The time this request was created	
tag	str	A custom tag for this request	
properties	IOrderProperties	The order properties for this request	None

Examples

Demonstration Algorithms
MarginCallEventsAlgorithm.py Python

Scheduled Events

Introduction

Scheduled Events let you trigger code to run at specific times of day, regardless of your algorithm's data subscriptions. It's easier and more reliable to execute time-based events with Scheduled Events than checking the current algorithm time in the `OnData` event handler.

Create Scheduled Events

To create a Scheduled Event, call the `Schedule.On` method. The method expects a `DateRules` object, a `TimeRules` object, and a function to execute. The following examples demonstrate some common Scheduled Events.

Schedule Events Before Market Open

You may want to train a model or fetch historical data before the market opens. The following example demonstrates how to set a Scheduled Event for 10 minutes before the market opens.

```
PY
self.Schedule.On(self.DateRules.EveryDay("SPY"),
                 self.TimeRules.AfterMarketOpen("SPY", -10),
                 self.BeforeMarketOpen)
```

Schedule Events on the Last Trading Day of the Week

You may want to rebalance your portfolio on the last trading day of each week and factor in market holidays. The following example demonstrates how to set a Scheduled Event for the last trading day of each week 30 minutes before the market closes.

```
PY
self.Schedule.On(self.DateRules.WeekEnd("SPY"),
                 self.TimeRules.BeforeMarketClose("SPY", 30),
                 self.Rebalance)
```

Schedule Events on Regular Intervals Throughout the Trading Day

You may want to perform some action on a regular interval through each trading day. The following example demonstrates how to set a Scheduled Event for every 30 minutes through the trading day for SPY.

```
PY
self.Schedule.On(self.DateRules.EveryDay("SPY"),
                 self.TimeRules.Every(timedelta(minutes=30)),
                 self.SomeAction)
```

Date Rules

The following table describes the supported `DateRules` :

Member	Description
`self.DateRules.SetDefaultTimeZone(time_zone: DateTimeZone)`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`self.DateRules.On(year: int, month: int, day: int)`	Trigger an event on a specific date.
`self. DateRules.EveryDay()`	Trigger an event every day.
`self.DateRules.EveryDay(symbol: Symbol)`	Trigger an event every day a specific symbol is trading.
`self.DateRules.Every(days: List[DayOfWeek])`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`self.DateRules.MonthStart(daysOffset: int = 0)`	Trigger an event on the first day of each month plus an offset.
`self.DateRules.MonthStart(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`self.DateRules.MonthEnd(daysOffset: int = 0)`	Trigger an event on the last day of each month minus an offset.
`self.DateRules.MonthEnd(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`self.DateRules.WeekStart(daysOffset: int = 0)`	Trigger an event on the first day of each week plus an offset.
`self.DateRules.WeekStart(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`self.DateRules.WeekEnd(daysOffset: int = 0)`	Trigger an event on the last day of each week minus an offset.
`self.DateRules.WeekEnd(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`self. DateRules.Today`	Trigger an event once today.
`self. DateRules.Tomorrow`	Trigger an event once tomorrow.

Time Rules

The following table describes the supported `TimeRules` :

Member	Description
`self.TimeRules.SetDefaultTimeZone(time_zone: DateTimeZone)`	Sets the time zone for the `TimeRules` object used in all methods in this table, except when a different time zone is given. The default time zone is the algorithm time zone .
`self.TimeRules.AfterMarketOpen(symbol: Symbol, minutesAfterOpen: float = 0, extendedMarketOpen: bool = False)`	Trigger an event a few minutes after market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`self.TimeRules.BeforeMarketClose(symbol: Symbol, minutesBeforeClose: float = 0, extendedMarketOpen: bool = False)`	Trigger an event a few minutes before market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`self.TimeRules.Every(interval: timedelta)`	Trigger an event every period interval starting at midnight.
`self. TimeRules.Now`	Trigger an event at the current time of day.
`self. TimeRules.Midnight`	Trigger an event at midnight.
`self. TimeRules.Noon`	Trigger an event at noon.
`self.TimeRules.At(hour: int, minute: int, second: int = 0)`	Trigger an event at a specific time of day (e.g. 13:10).
`self.TimeRules.At(hour: int, minute: int, second: int, time_zone: DateTimeZone)`	Trigger an event at a specific time of day in the given time zone (e.g. 13:10 UTC).

Remove Scheduled Events

If you no longer need a Scheduled Event in your algorithm, remove it so your algorithm doesn't execute unnecessary functions. To remove a Scheduled Event, save a reference to the Scheduled Event when you create it and then call the `Remove` method to remove it.

```
# Create a Scheduled Event
scheduled_event = self.Schedule.On(self.DateRules.EveryDay("SPY"),
                                   self.TimeRules.AfterMarketOpen("SPY", 10),
                                   self.ten_minutes_after_open)

# Remove the Scheduled Event
self.Schedule.Remove(scheduled_event)
```

Common Errors

Common errors with Scheduled Events include stale fills and execution timeouts.

Stale Fills

A common error is to subscribe to daily resolution data and set a Scheduled Event intraday to place trades. If you trade

intraday with daily data, you get stale fills.

Another common error is to set a Scheduled Event to trade immediately after the market open on illiquid securities. Illiquid securities can have no trades for the first few minutes after the market open. If you trade during this time, you get stale fills.

Execution Timeouts

If your Scheduled Event takes longer than 10 minutes to execute, your algorithm will timeout. To increase the amount of time that your Scheduled Event can run, replace your Scheduled Event with a training session .

Execution Sequence

The algorithm manager calls events in the following order:

1. Scheduled Events
2. Consolidation event handlers
3. `OnData` event handler

This event flow is important to note. For instance, if your consolidation handlers or `OnData` event handler appends data to a `RollingWindow` and you use that `RollingWindow` in your Scheduled Event, when the Scheduled Event executes, the `RollingWindow` won't contain the most recent data.

Live Trading Considerations

In live trading, Scheduled Events execute in a parallel thread based on a real-time clock. If you set a Scheduled Event to fire at 10:00 AM, it executes at exactly 10:00 AM. In backtesting, Scheduled Events are part of the main algorithm manager loop, so they may not execute exactly when you set them. For example, if your algorithm subscribes to minute resolution US Equity data with regular trading hours and you set a Scheduled Event to occur at 2:00 AM, your Scheduled Event will execute at 9:31 AM when the next bar is fed into your algorithm.

The difference between live trading and backtesting is important to note because it can affect your algorithm's behavior. There are two common scenarios to consider.

Execution Timing and Backtest Timeouts

Take the following scenario:

- You set Scheduled Events for 2:00 AM, 3:00 AM, and 4:00 AM each day.
- Each Scheduled Event takes eight minutes to execute.
- Your algorithm only subscribes to US Equity securities without extended market hours (9:30 AM - 4:00 PM).

In this scenario, the Scheduled Events each fire at the correct time and execute without error in live trading. In backtesting, all of the Scheduled Events execute at 9:31 AM when your algorithm receives the first bar of the trading day. Since all the Scheduled Events take eight minutes to execute, the algorithm tries to execute all the Scheduled Events but reaches the 10-minute timeout and the backtest stops execution.

Live Data Delays

In backtests, your algorithm receives data at perfect timing. If you request minute resolution data, your algorithm receives the bars at the top of each minute. In live trading, bars have a slight delay, so you may receive them

milliseconds after the top of each minute. Take the following scenario:

- You subscribe to minute resolution data
- You set a Scheduled Event for 10:00 AM
- The Scheduled Event checks the current asset price

In live trading, the Scheduled Event executes at exactly 10:00 AM but your algorithm may receive the 9:59-10:00 AM bar at 10:00:00.01 AM. Therefore, when you check the price in the Scheduled Event, the price from the 9:58-9:59 AM bar is the latest price. In backtesting, the Scheduled Event gets the price from the 9:59-10:00 AM bar since your algorithm receives the bar at perfect timing.

Examples

PY

```
# Schedule an event to fire at a specific date/time
self.Schedule.On(self.DateRules.On(2013, 10, 7),
                 self.TimeRules.At(13, 0),
                 lambda: self.Log(f"SpecificTime: Fired at : {self.Time}"))

# Schedule an event to fire every trading day for a security the
# The time rule here tells it to fire at 13:00:00 UTC
self.Schedule.On(self.DateRules.EveryDay("SPY"),
                 self.TimeRules.At(13, 0, 0, TimeZones.Utc),
                 lambda: self.Log(f"EveryDay.SPY SpecificTime: Fired at: {self.Time}"))

# Schedule an event to fire every trading day for a security the
# The time rule here tells it to fire 10 minutes after SPY's market open
self.Schedule.On(self.DateRules.EveryDay("SPY"),
                 self.TimeRules.AfterMarketOpen("SPY", 10),
                 lambda: self.Log(f"EveryDay.SPY 10 min after open: Fired at: {self.Time}"))

# Schedule an event to fire every trading day for a security the
# The time rule here tells it to fire 10 minutes before SPY's market close
self.Schedule.On(self.DateRules.EveryDay("SPY"),
                 self.TimeRules.BeforeMarketClose("SPY", 10),
                 lambda: self.Log(f"EveryDay.SPY 10 min before close: Fired at: {self.Time}"))

# Schedule an event to fire on certain days of the week
self.Schedule.On(self.DateRules.Every(DayOfWeek.Monday, DayOfWeek.Friday),
                 self.TimeRules.At(12, 0),
                 lambda: self.Log(f"Mon/Fri at 12pm: Fired at: {self.Time}"))

# Schedule an event to fire once today at when this method is called (now)
self.Schedule.On(self.DateRules.Today,
                 self.TimeRules.Now,
                 lambda: self.Log(f"Now: Fired at: {self.Time}"))

# Schedule an event to fire once tomorrow at midnight
self.Schedule.On(self.DateRules.Tomorrow,
                 self.TimeRules.Midnight,
                 lambda: self.Log(f"Tomorrow at midnight: Fired at: {self.Time}"))

# Schedule an event to fire once today at noon
self.Schedule.On(self.DateRules.Today,
                 self.TimeRules.Noon,
                 lambda: self.Log(f"Today at noon: Fired at: {self.Time}"))

# the scheduling methods return the ScheduledEvent object which can be used
# for other things here I set the event up to check the portfolio value every
# 10 minutes, and liquidate if we have too many losses
self.Schedule.On(self.DateRules.EveryDay(),
                 self.TimeRules.Every(timedelta(minutes=10)),
                 self.LiquidateUnrealizedLosses)

# Schedule an event to fire at the beginning of the month, the symbol is optional.
# if specified, it will fire the first trading day for that symbol of the month,
# if not specified it will fire on the first day of the month
self.Schedule.On(self.DateRules.MonthStart("SPY"),
                 self.TimeRules.AfterMarketOpen("SPY"),
                 self.RebalancingCode)

# Schedule an event to fire at the end of the month, the symbol is optional.
```

```
    # if specified, it will fire the last trading day for that symbol of the month,
    # if not specified it will fire on the first day of the month
    self.Schedule.On(self.DateRules.MonthEnd("SPY"),
                     self.TimeRules.BeforeMarketClose("SPY"),
                     self.RebalancingCode)

    # Schedule an event to fire at the beginning of the week, the symbol is optional.
    # if specified, it will fire the first trading day for that symbol of the week,
    # if not specified it will fire on the first day of the week
    self.Schedule.On(self.DateRules.WeekStart("SPY"),
                     self.TimeRules.AfterMarketOpen("SPY", 5),
                     self.RebalancingCode)

    # Schedule an event to fire at the end of the week, the symbol is optional.
    # if specified, it will fire the last trading day for that symbol of the week,
    # if not specified it will fire on the first day of the week
    self.Schedule.On(self.DateRules.WeekEnd("SPY"),
                     self.TimeRules.BeforeMarketClose("SPY", 5),
                     self.RebalancingCode)


    # The following methods are not defined in Initialize:
    def LiquidateUnrealizedLosses(self) -> None:
        ''' if we have over 1000 dollars in unrealized losses, liquidate'''
        if self.Portfolio.TotalUnrealizedProfit < -1000:
            self.Log(f"Liquidated due to unrealized losses at: {self.Time}")
            self.Liquidate()

    def RebalancingCode(self) -> None:
        ''' Good spot for rebalancing code?'''
        pass
```

Demonstration Algorithm
ScheduledEventsAlgorithm.py Python

Indicators

Indicators

Supported Indicators

Indicators translate a stream of data points into a numerical value you can use to detect trading opportunities. LEAN provides more than 100 pre-built technical indicators and candlestick patterns you can use in your algorithms. You can use any of the following indicators. Click one to learn more.

Candlestick Patterns

Absolute Price Oscillator

Acceleration Bands

Accumulation Distribution

Accumulation Distribution Oscillator

Advance Decline Difference

Advance Decline Ratio

Advance Decline Volume Ratio

Arms Index

Arnaud Legoux Moving Average

Aroon Oscillator

Augen Price Spike

Auto Regressive Integrated Moving Average

Average Directional Index

Average Directional Movement Index Rating

Average True Range

Awesome Oscillator

Balance Of Power

Beta

Bollinger Bands

Chaikin Money Flow

Chande Momentum Oscillator

Commodity Channel Index

Coppock Curve

De Marker Indicator

Delay

Detrended Price Oscillator

Donchian Channel

Double Exponential Moving Average

Ease Of Movement Value

Exponential Moving Average

Filtered Identity

Fisher Transform

Fractal Adaptive Moving Average

Heikin Ashi

Hilbert Transform

Hull Moving Average

Ichimoku Kinko Hyo

Identity

Intraday Vwap

Kaufman Adaptive Moving Average

Kaufman Efficiency Ratio

Keltner Channels

Least Squares Moving Average

Linear Weighted Moving Average

Log Return

Mass Index

Maximum

Mc Clellan Oscillator

Mc Clellan Summation Index

Mean Absolute Deviation

Mid Point

Mid Price

Minimum

Momentum

Momentum Percent

Momersion Indicator

Money Flow Index

Moving Average Convergence Divergence

Normalized Average True Range

On Balance Volume

Parabolic Stop And Reverse

Percentage Price Oscillator

Pivot Points High Low

Rate Of Change

Rate Of Change Percent

Rate Of Change Ratio

Regression Channel

Relative Daily Volume

Relative Moving Average

Relative Strength Index

Relative Vigor Index

Schaff Trend Cycle

Sharpe Ratio

Simple Moving Average

Sortino Ratio

Standard Deviation

Stochastic

Sum

Super Trend

Swiss Army Knife

T3 Moving Average

Target Downside Deviation

Time Profile

Triangular Moving Average

Triple Exponential Moving Average

Trix

True Range

True Strength Index

Ultimate Oscillator

Variance

Volume Profile

Volume Weighted Average Price Indicator

Wilder Accumulative Swing Index

Wilder Moving Average

Wilder Swing Index

Williams Percent R

Supported Indicators

Candlestick Patterns

You can use any of the following candlestick patterns. Click one to learn more.

Abandoned Baby

Advance Block

Belt Hold

Breakaway

Closing Marubozu

Concealed Baby Swallow

Counterattack

Dark Cloud Cover

Doji

Doji Star

Dragonfly Doji

Engulfing

Evening Doji Star

Evening Star

Gap Side By Side White

Gravestone Doji

Hammer

Hanging Man

Harami

Harami Cross

High Wave Candle

Hikkake

Hikkake Modified

Homing Pigeon

Identical Three Crows

In Neck

Inverted Hammer

Kicking

Kicking By Length

Ladder Bottom

Long Legged Doji

Long Line Candle

Marubozu

Mat Hold

Matching Low

Morning Doji Star

Morning Star

On Neck

Piercing

Rickshaw Man

Rise Fall Three Methods

Separating Lines

Shooting Star

Short Line Candle

Spinning Top

Stalled Pattern

Stick Sandwich

Takuri

Tasuki Gap

Three Black Crows

Three Inside

Three Line Strike

Three Outside

Three Stars In South

Three White Soldiers

Thrusting

Tristar

Two Crows

Unique Three River

Up Down Gap Three Methods

Upside Gap Two Crows

Supported Indicators

Abandoned Baby

Introduction

Create a new Abandoned Baby candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using AbandonedBaby Indicator

To create an automatic indicators for `AbandonedBaby` , call the `AbandonedBaby` helper method from the `QCAlgorithm` class. The `AbandonedBaby` method creates a `AbandonedBaby` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AbandonedBabyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.abandonedbaby = self.AbandonedBaby(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.abandonedbaby.IsReady:
            # The current value of self.abandonedbaby is represented by self.abandonedbaby.Current.Value
            self.Plot("AbandonedBaby", "abandonedbaby", self.abandonedbaby.Current.Value)
```

The following reference table describes the `AbandonedBaby` method:

INDICATORS

AbandonedBaby() 1/1

```
AbandonedBaby QuantConnect.Algorithm.CandlestickPatterns.AbandonedBaby (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `AbandonedBaby` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`AbandonedBaby` - The new `AbandonedBaby` indicator object.

Definition at line 161 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AbandonedBaby` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class AbandonedBabyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.abandonedbaby = AbandonedBaby()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.abandonedbaby.Update(bar)

        if self.abandonedbaby.IsReady:
            # The current value of self.abandonedbaby is represented by self.abandonedbaby.Current.Value
            self.Plot("AbandonedBaby", "abandonedbaby", self.abandonedbaby.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AbandonedBabyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.abandonedbaby = AbandonedBaby()
        self.RegisterIndicator(self.symbol, self.abandonedbaby, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.abandonedbaby.IsReady:
            # The current value of self.abandonedbaby is represented by self.abandonedbaby.Current.Value
            self.Plot("AbandonedBaby", "abandonedbaby", self.abandonedbaby.Current.Value)
```

The following reference table describes the `AbandonedBaby` constructor:

INDICATORS

AbandonedBaby() 1/3

```
AbandonedBaby QuantConnect.Indicators.CandlestickPatterns.AbandonedBaby (
string name,
*decimal penetration
)
```

Initializes a new instance of the `AbandonedBaby` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	penetration	*(Optional)* Percentage of penetration of a candle within another candle. Default: 0.3m.

Return

`AbandonedBaby` - The new `AbandonedBaby` indicator object.

Definition at line 57 of file Indicators/CandlestickPatterns/AbandonedBaby.cs.

INDICATORS

AbandonedBaby() 2/3

```
AbandonedBaby QuantConnect.Indicators.CandlestickPatterns.AbandonedBaby (
decimal penetration
)
```

Initializes a new instance of the `AbandonedBaby` class.

Show Details

Parameters		
`decimal`	penetration	Percentage of penetration of a candle within another candle.

Return

`AbandonedBaby` - The new `AbandonedBaby` indicator object.

Definition at line 72 of file Indicators/CandlestickPatterns/AbandonedBaby.cs.

INDICATORS

AbandonedBaby() 3/3

```
AbandonedBaby QuantConnect.Indicators.CandlestickPatterns.AbandonedBaby (
)
```

Initializes a new instance of the `AbandonedBaby` class.

Show Details

This method requires no argument input.

Return

`AbandonedBaby` - The new `AbandonedBaby` indicator object.

Definition at line 80 of file Indicators/CandlestickPatterns/AbandonedBaby.cs.

Supported Indicators

Advance Block

Introduction

Create a new Advance Block candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using AdvanceBlock Indicator

To create an automatic indicators for `AdvanceBlock` , call the `AdvanceBlock` helper method from the `QCAlgorithm` class. The `AdvanceBlock` method creates a `AdvanceBlock` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AdvanceBlockAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.advanceblock = self.AdvanceBlock(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.advanceblock.IsReady:
            # The current value of self.advanceblock is represented by self.advanceblock.Current.Value
            self.Plot("AdvanceBlock", "advanceblock", self.advanceblock.Current.Value)
```

The following reference table describes the `AdvanceBlock` method:

INDICATORS

AdvanceBlock() 1/1

```
AdvanceBlock QuantConnect.Algorithm.CandlestickPatterns.AdvanceBlock (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `AdvanceBlock` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`AdvanceBlock` - The new `AdvanceBlock` indicator object.

Definition at line 177 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AdvanceBlock` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class AdvanceBlockAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.advanceblock = AdvanceBlock()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.advanceblock.Update(bar)

        if self.advanceblock.IsReady:
            # The current value of self.advanceblock is represented by self.advanceblock.Current.Value
            self.Plot("AdvanceBlock", "advanceblock", self.advanceblock.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class AdvanceBlockAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.advanceblock = AdvanceBlock()
        self.RegisterIndicator(self.symbol, self.advanceblock, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.advanceblock.IsReady:
            # The current value of self.advanceblock is represented by self.advanceblock.Current.Value
            self.Plot("AdvanceBlock", "advanceblock", self.advanceblock.Current.Value)
```

The following reference table describes the `AdvanceBlock` constructor:

INDICATORS

AdvanceBlock() 1/2

```
AdvanceBlock QuantConnect.Indicators.CandlestickPatterns.AdvanceBlock (
string name
)
```

Initializes a new instance of the `AdvanceBloc` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`AdvanceBlock` - The new `AdvanceBlock` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/AdvanceBlock.cs.

INDICATORS

AdvanceBlock() 2/2

```
AdvanceBlock QuantConnect.Indicators.CandlestickPatterns.AdvanceBlock (

)
```

Initializes a new instance of the `AdvanceBloc` class.

Show Details

This method requires no argument input.

Return

`AdvanceBlock` - The new `AdvanceBlock` indicator object.

Definition at line 70 of file Indicators/CandlestickPatterns/AdvanceBlock.cs.

Supported Indicators

Belt Hold

Introduction

Create a new Belt-hold candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using BeltHold Indicator

To create an automatic indicators for `BeltHold` , call the `BeltHold` helper method from the `QCAlgorithm` class. The `BeltHold` method creates a `BeltHold` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class BeltHoldAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.belthold = self.BeltHold(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.belthold.IsReady:
            # The current value of self.belthold is represented by self.belthold.Current.Value
            self.Plot("BeltHold", "belthold", self.belthold.Current.Value)
```

The following reference table describes the `BeltHold` method:

INDICATORS

BeltHold() 1/1

```
BeltHold QuantConnect.Algorithm.CandlestickPatterns.BeltHold (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `BeltHold` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`BeltHold` - The new `BeltHold` indicator object.

Definition at line 193 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `BeltHold` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class BeltHoldAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.belthold = BeltHold()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.belthold.Update(bar)

        if self.belthold.IsReady:
            # The current value of self.belthold is represented by self.belthold.Current.Value
            self.Plot("BeltHold", "belthold", self.belthold.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class BeltHoldAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.belthold = BeltHold()
        self.RegisterIndicator(self.symbol, self.belthold, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.belthold.IsReady:
            # The current value of self.belthold is represented by self.belthold.Current.Value
            self.Plot("BeltHold", "belthold", self.belthold.Current.Value)
```

The following reference table describes the `BeltHold` constructor:

INDICATORS

BeltHold() 1/2

```
BeltHold QuantConnect.Indicators.CandlestickPatterns.BeltHold (
string name
)
```

Initializes a new instance of the `BeltHol` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`BeltHold` - The new `BeltHold` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/BeltHold.cs.

INDICATORS

BeltHold() 2/2

```
BeltHold QuantConnect.Indicators.CandlestickPatterns.BeltHold (

)
```

Initializes a new instance of the `BeltHol` class.

Show Details

This method requires no argument input.

Return

`BeltHold` - The new `BeltHold` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/BeltHold.cs.

Supported Indicators

Breakaway

Introduction

Create a new Breakaway candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Breakaway Indicator

To create an automatic indicators for `Breakaway` , call the `Breakaway` helper method from the `QCAlgorithm` class. The `Breakaway` method creates a `Breakaway` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class BreakawayAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.breakaway = self.Breakaway(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.breakaway.IsReady:
            # The current value of self.breakaway is represented by self.breakaway.Current.Value
            self.Plot("Breakaway", "breakaway", self.breakaway.Current.Value)
```

The following reference table describes the `Breakaway` method:

INDICATORS

Breakaway() 1/1

```
Breakaway QuantConnect.Algorithm.CandlestickPatterns.Breakaway (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Breakaway` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Breakaway` - The new `Breakaway` indicator object.

Definition at line 209 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Breakaway` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class BreakawayAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.breakaway = Breakaway()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.breakaway.Update(bar)

        if self.breakaway.IsReady:
            # The current value of self.breakaway is represented by self.breakaway.Current.Value
            self.Plot("Breakaway", "breakaway", self.breakaway.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class BreakawayAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.breakaway = Breakaway()
        self.RegisterIndicator(self.symbol, self.breakaway, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.breakaway.IsReady:
            # The current value of self.breakaway is represented by self.breakaway.Current.Value
            self.Plot("Breakaway", "breakaway", self.breakaway.Current.Value)
```

The following reference table describes the `Breakaway` constructor:

INDICATORS

Breakaway() 1/2

```
Breakaway QuantConnect.Indicators.CandlestickPatterns.Breakaway (
string name
)
```

Initializes a new instance of the `Breakawa` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`Breakaway` - The new `Breakaway` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/Breakaway.cs.

INDICATORS

Breakaway() 2/2

```
Breakaway QuantConnect.Indicators.CandlestickPatterns.Breakaway (

)
```

Initializes a new instance of the `Breakawa` class.

Show Details

This method requires no argument input.

Return

`Breakaway` - The new `Breakaway` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/Breakaway.cs.

Supported Indicators

Closing Marubozu

Introduction

Create a new Closing Marubozu candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ClosingMarubozu Indicator

To create an automatic indicators for `ClosingMarubozu` , call the `ClosingMarubozu` helper method from the `QCAlgorithm` class. The `ClosingMarubozu` method creates a `ClosingMarubozu` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ClosingMarubozuAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.closingmarubozu = self.ClosingMarubozu(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.closingmarubozu.IsReady:
            # The current value of self.closingmarubozu is represented by
self.closingmarubozu.Current.Value
            self.Plot("ClosingMarubozu", "closingmarubozu", self.closingmarubozu.Current.Value)
```

The following reference table describes the `ClosingMarubozu` method:

INDICATORS

ClosingMarubozu() 1/1

```
ClosingMarubozu QuantConnect.Algorithm.CandlestickPatterns.ClosingMarubozu (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ClosingMarubozu` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ClosingMarubozu` - The new `ClosingMarubozu` indicator object.

Definition at line 225 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ClosingMarubozu` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ClosingMarubozuAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.closingmarubozu = ClosingMarubozu()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.closingmarubozu.Update(bar)

        if self.closingmarubozu.IsReady:
            # The current value of self.closingmarubozu is represented by
self.closingmarubozu.Current.Value
            self.Plot("ClosingMarubozu", "closingmarubozu", self.closingmarubozu.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ClosingMarubozuAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.closingmarubozu = ClosingMarubozu()
        self.RegisterIndicator(self.symbol, self.closingmarubozu, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.closingmarubozu.IsReady:
            # The current value of self.closingmarubozu is represented by
self.closingmarubozu.Current.Value
            self.Plot("ClosingMarubozu", "closingmarubozu", self.closingmarubozu.Current.Value)
```

The following reference table describes the `ClosingMarubozu` constructor:

INDICATORS

ClosingMarubozu() 1/2

```
ClosingMarubozu QuantConnect.Indicators.CandlestickPatterns.ClosingMarubozu (
string name
)
```

Initializes a new instance of the `ClosingMaruboz` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`ClosingMarubozu` - The new `ClosingMarubozu` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/ClosingMarubozu.cs.

INDICATORS

ClosingMarubozu() 2/2

```
ClosingMarubozu QuantConnect.Indicators.CandlestickPatterns.ClosingMarubozu (
)
```

Initializes a new instance of the `ClosingMaruboz` class.

Show Details

This method requires no argument input.

Return

`ClosingMarubozu` - The new `ClosingMarubozu` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/ClosingMarubozu.cs.

Supported Indicators

Concealed Baby Swallow

Introduction

Create a new Concealed Baby Swallow candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ConcealedBabySwallow Indicator

To create an automatic indicators for `ConcealedBabySwallow` , call the `ConcealedBabySwallow` helper method from the `QCAlgorithm` class. The `ConcealedBabySwallow` method creates a `ConcealedBabySwallow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ConcealedBabySwallowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.concealedbabyswallow = self.ConcealedBabySwallow(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.concealedbabyswallow.IsReady:
            # The current value of self.concealedbabyswallow is represented by
self.concealedbabyswallow.Current.Value
            self.Plot("ConcealedBabySwallow", "concealedbabyswallow",
self.concealedbabyswallow.Current.Value)
```

The following reference table describes the `ConcealedBabySwallow` method:

INDICATORS

ConcealedBabySwallow() 1/1

```
ConcealedBabySwallow QuantConnect.Algorithm.CandlestickPatterns.ConcealedBabySwallow (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ConcealedBabySwallow` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ConcealedBabySwallow` - The new `ConcealedBabySwallow` indicator object.

Definition at line 241 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ConcealedBabySwallow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ConcealedBabySwallowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.concealedbabyswallow = ConcealedBabySwallow()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.concealedbabyswallow.Update(bar)

        if self.concealedbabyswallow.IsReady:
            # The current value of self.concealedbabyswallow is represented by
self.concealedbabyswallow.Current.Value
            self.Plot("ConcealedBabySwallow", "concealedbabyswallow",
self.concealedbabyswallow.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ConcealedBabySwallowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.concealedbabyswallow = ConcealedBabySwallow()
        self.RegisterIndicator(self.symbol, self.concealedbabyswallow, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.concealedbabyswallow.IsReady:
            # The current value of self.concealedbabyswallow is represented by
self.concealedbabyswallow.Current.Value
            self.Plot("ConcealedBabySwallow", "concealedbabyswallow",
self.concealedbabyswallow.Current.Value)
```

The following reference table describes the `ConcealedBabySwallow` constructor:

INDICATORS

ConcealedBabySwallow() 1/2

```
ConcealedBabySwallow QuantConnect.Indicators.CandlestickPatterns.ConcealedBabySwallow (
string name
)
```

Initializes a new instance of the `ConcealedBabySwallo` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`ConcealedBabySwallow` - The new `ConcealedBabySwallow` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/ConcealedBabySwallow.cs.

INDICATORS

ConcealedBabySwallow() 2/2

```
ConcealedBabySwallow QuantConnect.Indicators.CandlestickPatterns.ConcealedBabySwallow (

)
```

Initializes a new instance of the `ConcealedBabySwallo` class.

Show Details

This method requires no argument input.

Return

`ConcealedBabySwallow` - The new `ConcealedBabySwallow` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/ConcealedBabySwallow.cs.

Supported Indicators

Counterattack

Introduction

Create a new Counterattack candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Counterattack Indicator

To create an automatic indicators for `Counterattack` , call the `Counterattack` helper method from the `QCAlgorithm` class. The `Counterattack` method creates a `Counterattack` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class CounterattackAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.counterattack = self.Counterattack(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.counterattack.IsReady:
            # The current value of self.counterattack is represented by self.counterattack.Current.Value
            self.Plot("Counterattack", "counterattack", self.counterattack.Current.Value)
```

The following reference table describes the `Counterattack` method:

INDICATORS

Counterattack() 1/1

```
Counterattack QuantConnect.Algorithm.CandlestickPatterns.Counterattack (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Counterattack` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Counterattack` - The new `Counterattack` indicator object.

Definition at line 257 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Counterattack` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class CounterattackAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.counterattack = Counterattack()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.counterattack.Update(bar)

        if self.counterattack.IsReady:
            # The current value of self.counterattack is represented by self.counterattack.Current.Value
            self.Plot("Counterattack", "counterattack", self.counterattack.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class CounterattackAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.counterattack = Counterattack()
        self.RegisterIndicator(self.symbol, self.counterattack, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.counterattack.IsReady:
            # The current value of self.counterattack is represented by self.counterattack.Current.Value
            self.Plot("Counterattack", "counterattack", self.counterattack.Current.Value)
```

The following reference table describes the `Counterattack` constructor:

INDICATORS

Counterattack() 1/2

```
Counterattack QuantConnect.Indicators.CandlestickPatterns.Counterattack (
string name
)
```

Initializes a new instance of the `Counterattac` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Counterattack` - The new `Counterattack` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/Counterattack.cs.

INDICATORS

Counterattack() 2/2

```
Counterattack QuantConnect.Indicators.CandlestickPatterns.Counterattack (

)
```

Initializes a new instance of the `Counterattac` class.

Show Details

This method requires no argument input.

Return

`Counterattack` - The new `Counterattack` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/Counterattack.cs.

Supported Indicators

Dark Cloud Cover

Introduction

Create a new Dark Cloud Cover candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using DarkCloudCover Indicator

To create an automatic indicators for `DarkCloudCover` , call the `DarkCloudCover` helper method from the `QCAlgorithm` class. The `DarkCloudCover` method creates a `DarkCloudCover` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class DarkCloudCoverAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.darkcloudcover = self.DarkCloudCover(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.darkcloudcover.IsReady:
            # The current value of self.darkcloudcover is represented by self.darkcloudcover.Current.Value
            self.Plot("DarkCloudCover", "darkcloudcover", self.darkcloudcover.Current.Value)
```

The following reference table describes the `DarkCloudCover` method:

INDICATORS

DarkCloudCover() 1/1

```
DarkCloudCover QuantConnect.Algorithm.CandlestickPatterns.DarkCloudCover (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `DarkCloudCover` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`DarkCloudCover` - The new `DarkCloudCover` indicator object.

Definition at line 274 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `DarkCloudCover` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class DarkCloudCoverAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.darkcloudcover = DarkCloudCover()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.darkcloudcover.Update(bar)

        if self.darkcloudcover.IsReady:
            # The current value of self.darkcloudcover is represented by self.darkcloudcover.Current.Value
            self.Plot("DarkCloudCover", "darkcloudcover", self.darkcloudcover.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class DarkCloudCoverAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.darkcloudcover = DarkCloudCover()
        self.RegisterIndicator(self.symbol, self.darkcloudcover, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.darkcloudcover.IsReady:
            # The current value of self.darkcloudcover is represented by self.darkcloudcover.Current.Value
            self.Plot("DarkCloudCover", "darkcloudcover", self.darkcloudcover.Current.Value)
```

The following reference table describes the `DarkCloudCover` constructor:

INDICATORS

DarkCloudCover() 1/3

```
DarkCloudCover QuantConnect.Indicators.CandlestickPatterns.DarkCloudCover (
string name,
*decimal penetration
)
```

Initializes a new instance of the `DarkCloudCove` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	penetration	*(Optional) (Optional)* Percentage of penetration of a candle within another candle. Default: 0.5m.

Return

`DarkCloudCover` - The new `DarkCloudCover` indicator object.

Definition at line 48 of file Indicators/CandlestickPatterns/DarkCloudCover.cs.

INDICATORS

DarkCloudCover() 2/3

```
DarkCloudCover QuantConnect.Indicators.CandlestickPatterns.DarkCloudCover (
decimal penetration
)
```

Initializes a new instance of the `DarkCloudCove` class.

Show Details

Parameters		
`decimal`	penetration	Percentage of penetration of a candle within another candle.

Return

`DarkCloudCover` - The new `DarkCloudCover` indicator object.

Definition at line 60 of file Indicators/CandlestickPatterns/DarkCloudCover.cs.

INDICATORS

DarkCloudCover() 3/3

```
DarkCloudCover QuantConnect.Indicators.CandlestickPatterns.DarkCloudCover (
)
```

Initializes a new instance of the `DarkCloudCove` class.

Show Details

This method requires no argument input.

Return

`DarkCloudCover` - The new `DarkCloudCover` indicator object.

Definition at line 68 of file Indicators/CandlestickPatterns/DarkCloudCover.cs.

Supported Indicators

Doji

Introduction

Create a new Doji candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Doji Indicator

To create an automatic indicators for `Doji` , call the `Doji` helper method from the `QCAlgorithm` class. The `Doji` method creates a `Doji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class DojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.doji = self.Doji(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.doji.IsReady:
            # The current value of self.doji is represented by self.doji.Current.Value
            self.Plot("Doji", "doji", self.doji.Current.Value)
```

The following reference table describes the `Doji` method:

INDICATORS

Doji() 1/1

```
Doji QuantConnect.Algorithm.CandlestickPatterns.Doji (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Doji` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Doji` - The new `Doji` indicator object.

Definition at line 290 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Doji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class DojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.doji = Doji()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.doji.Update(bar)

        if self.doji.IsReady:
            # The current value of self.doji is represented by self.doji.Current.Value
            self.Plot("Doji", "doji", self.doji.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class DojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.doji = Doji()
        self.RegisterIndicator(self.symbol, self.doji, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.doji.IsReady:
            # The current value of self.doji is represented by self.doji.Current.Value
            self.Plot("Doji", "doji", self.doji.Current.Value)
```

The following reference table describes the `Doji` constructor:

INDICATORS

Doji() 1/2

```
Doji QuantConnect.Indicators.CandlestickPatterns.Doji (
string name
)
```

Initializes a new instance of the `Doj` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`Doji` - The new `Doji` indicator object.

Definition at line 40 of file Indicators/CandlestickPatterns/Doji.cs.

INDICATORS

Doji() 2/2

```
Doji QuantConnect.Indicators.CandlestickPatterns.Doji (
)
```

Initializes a new instance of the `Doj` class.

Show Details

This method requires no argument input.

Return

`Doji` - The new `Doji` indicator object.

Definition at line 49 of file Indicators/CandlestickPatterns/Doji.cs.

Supported Indicators

Doji Star

Introduction

Create a new Doji Star candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using DojiStar Indicator

To create an automatic indicators for `DojiStar` , call the `DojiStar` helper method from the `QCAlgorithm` class. The `DojiStar` method creates a `DojiStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class DojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dojistar = self.DojiStar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.dojistar.IsReady:
            # The current value of self.dojistar is represented by self.dojistar.Current.Value
            self.Plot("DojiStar", "dojistar", self.dojistar.Current.Value)
```

The following reference table describes the `DojiStar` method:

INDICATORS

DojiStar() 1/1

```
DojiStar QuantConnect.Algorithm.CandlestickPatterns.DojiStar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `DojiStar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`DojiStar` - The new `DojiStar` indicator object.

Definition at line 306 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `DojiStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class DojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dojistar = DojiStar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.dojistar.Update(bar)

        if self.dojistar.IsReady:
            # The current value of self.dojistar is represented by self.dojistar.Current.Value
            self.Plot("DojiStar", "dojistar", self.dojistar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class DojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dojistar = DojiStar()
        self.RegisterIndicator(self.symbol, self.dojistar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.dojistar.IsReady:
            # The current value of self.dojistar is represented by self.dojistar.Current.Value
            self.Plot("DojiStar", "dojistar", self.dojistar.Current.Value)
```

The following reference table describes the `DojiStar` constructor:

INDICATORS

DojiStar() 1/2

```
DojiStar QuantConnect.Indicators.CandlestickPatterns.DojiStar (
string name
)
```

Initializes a new instance of the `DojiSta` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`DojiStar` - The new `DojiStar` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/DojiStar.cs.

INDICATORS

DojiStar() 2/2

```
DojiStar QuantConnect.Indicators.CandlestickPatterns.DojiStar (

)
```

Initializes a new instance of the `DojiSta` class.

Show Details

This method requires no argument input.

Return

`DojiStar` - The new `DojiStar` indicator object.

Definition at line 57 of file Indicators/CandlestickPatterns/DojiStar.cs.

Supported Indicators

Dragonfly Doji

Introduction

Create a new Dragonfly Doji candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using DragonflyDoji Indicator

To create an automatic indicators for `DragonflyDoji` , call the `DragonflyDoji` helper method from the `QCAlgorithm` class. The `DragonflyDoji` method creates a `DragonflyDoji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class DragonflyDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dragonflydoji = self.DragonflyDoji(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.dragonflydoji.IsReady:
            # The current value of self.dragonflydoji is represented by self.dragonflydoji.Current.Value
            self.Plot("DragonflyDoji", "dragonflydoji", self.dragonflydoji.Current.Value)
```

The following reference table describes the `DragonflyDoji` method:

INDICATORS

DragonflyDoji() 1/1

```
DragonflyDoji QuantConnect.Algorithm.CandlestickPatterns.DragonflyDoji (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `DragonflyDoji` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`DragonflyDoji` - The new `DragonflyDoji` indicator object.

Definition at line 322 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `DragonflyDoji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class DragonflyDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dragonflydoji = DragonflyDoji()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.dragonflydoji.Update(bar)

        if self.dragonflydoji.IsReady:
            # The current value of self.dragonflydoji is represented by self.dragonflydoji.Current.Value
            self.Plot("DragonflyDoji", "dragonflydoji", self.dragonflydoji.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
PY
class DragonflyDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dragonflydoji = DragonflyDoji()
        self.RegisterIndicator(self.symbol, self.dragonflydoji, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.dragonflydoji.IsReady:
            # The current value of self.dragonflydoji is represented by self.dragonflydoji.Current.Value
            self.Plot("DragonflyDoji", "dragonflydoji", self.dragonflydoji.Current.Value)
```

The following reference table describes the `DragonflyDoji` constructor:

INDICATORS

DragonflyDoji() 1/2

```
DragonflyDoji QuantConnect.Indicators.CandlestickPatterns.DragonflyDoji (
string name
)
```

Initializes a new instance of the `DragonflyDoj` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`DragonflyDoji` - The new `DragonflyDoji` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/DragonflyDoji.cs.

INDICATORS

DragonflyDoji() 2/2

```
DragonflyDoji QuantConnect.Indicators.CandlestickPatterns.DragonflyDoji (
)
```

Initializes a new instance of the `DragonflyDoj` class.

Show Details

This method requires no argument input.

Return

`DragonflyDoji` - The new `DragonflyDoji` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/DragonflyDoji.cs.

Supported Indicators

Engulfing

Introduction

Create a new Engulfing candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Engulfing Indicator

To create an automatic indicators for `Engulfing` , call the `Engulfing` helper method from the `QCAlgorithm` class. The `Engulfing` method creates a `Engulfing` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class EngulfingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.engulfing = self.Engulfing(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.engulfing.IsReady:
            # The current value of self.engulfing is represented by self.engulfing.Current.Value
            self.Plot("Engulfing", "engulfing", self.engulfing.Current.Value)
```

The following reference table describes the `Engulfing` method:

INDICATORS

Engulfing() 1/1

```
Engulfing QuantConnect.Algorithm.CandlestickPatterns.Engulfing (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Engulfing` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Engulfing` - The new `Engulfing` indicator object.

Definition at line 338 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Engulfing` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class EngulfingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.engulfing = Engulfing()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.engulfing.Update(bar)

        if self.engulfing.IsReady:
            # The current value of self.engulfing is represented by self.engulfing.Current.Value
            self.Plot("Engulfing", "engulfing", self.engulfing.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class EngulfingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.engulfing = Engulfing()
        self.RegisterIndicator(self.symbol, self.engulfing, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.engulfing.IsReady:
            # The current value of self.engulfing is represented by self.engulfing.Current.Value
            self.Plot("Engulfing", "engulfing", self.engulfing.Current.Value)
```

The following reference table describes the `Engulfing` constructor:

INDICATORS

Engulfing() 1/2

```
Engulfing QuantConnect.Indicators.CandlestickPatterns.Engulfing (
string name
)
```

Initializes a new instance of the `Engulfin` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Engulfing` - The new `Engulfing` indicator object.

Definition at line 38 of file Indicators/CandlestickPatterns/Engulfing.cs.

INDICATORS

Engulfing() 2/2

```
Engulfing QuantConnect.Indicators.CandlestickPatterns.Engulfing (

)
```

Initializes a new instance of the `Engulfin` class.

Show Details

This method requires no argument input.

Return

`Engulfing` - The new `Engulfing` indicator object.

Definition at line 46 of file Indicators/CandlestickPatterns/Engulfing.cs.

Supported Indicators

Evening Doji Star

Introduction

Create a new Evening Doji Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using EveningDojiStar Indicator

To create an automatic indicators for `EveningDojiStar` , call the `EveningDojiStar` helper method from the `QCAlgorithm` class. The `EveningDojiStar` method creates a `EveningDojiStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class EveningDojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.eveningdojistar = self.EveningDojiStar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.eveningdojistar.IsReady:
            # The current value of self.eveningdojistar is represented by
self.eveningdojistar.Current.Value
            self.Plot("EveningDojiStar", "eveningdojistar", self.eveningdojistar.Current.Value)
```

The following reference table describes the `EveningDojiStar` method:

INDICATORS

EveningDojiStar() 1/1

```
EveningDojiStar QuantConnect.Algorithm.CandlestickPatterns.EveningDojiStar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `EveningDojiStar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`EveningDojiStar` - The new `EveningDojiStar` indicator object.

Definition at line 355 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `EveningDojiStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class EveningDojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.eveningdojistar = EveningDojiStar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.eveningdojistar.Update(bar)

        if self.eveningdojistar.IsReady:
            # The current value of self.eveningdojistar is represented by
self.eveningdojistar.Current.Value
            self.Plot("EveningDojiStar", "eveningdojistar", self.eveningdojistar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class EveningDojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.eveningdojistar = EveningDojiStar()
        self.RegisterIndicator(self.symbol, self.eveningdojistar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.eveningdojistar.IsReady:
            # The current value of self.eveningdojistar is represented by
self.eveningdojistar.Current.Value
            self.Plot("EveningDojiStar", "eveningdojistar", self.eveningdojistar.Current.Value)
```

The following reference table describes the `EveningDojiStar` constructor:

INDICATORS

EveningDojiStar() 1/3

```
EveningDojiStar QuantConnect.Indicators.CandlestickPatterns.EveningDojiStar (
string name,
*decimal penetration
)
```

Initializes a new instance of the `EveningDojiSta` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	penetration	*(Optional) (Optional)* Percentage of penetration of a candle within another candle. Default: 0.3m.

Return

`EveningDojiStar` - The new `EveningDojiStar` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/EveningDojiStar.cs.

INDICATORS

EveningDojiStar() 2/3

```
EveningDojiStar QuantConnect.Indicators.CandlestickPatterns.EveningDojiStar (
decimal penetration
)
```

Initializes a new instance of the `EveningDojiSta` class.

Show Details

Parameters		
`decimal`	penetration	Percentage of penetration of a candle within another candle.

Return

`EveningDojiStar` - The new `EveningDojiStar` indicator object.

Definition at line 70 of file Indicators/CandlestickPatterns/EveningDojiStar.cs.

INDICATORS

EveningDojiStar() 3/3

```
EveningDojiStar QuantConnect.Indicators.CandlestickPatterns.EveningDojiStar (
)
```

Initializes a new instance of the `EveningDojiSta` class.

Show Details

This method requires no argument input.

Return

`EveningDojiStar` - The new `EveningDojiStar` indicator object.

Definition at line 78 of file Indicators/CandlestickPatterns/EveningDojiStar.cs.

Supported Indicators

Evening Star

Introduction

Create a new Evening Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using EveningStar Indicator

To create an automatic indicators for `EveningStar` , call the `EveningStar` helper method from the `QCAlgorithm` class. The `EveningStar` method creates a `EveningStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class EveningStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.eveningstar = self.EveningStar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.eveningstar.IsReady:
            # The current value of self.eveningstar is represented by self.eveningstar.Current.Value
            self.Plot("EveningStar", "eveningstar", self.eveningstar.Current.Value)
```

The following reference table describes the `EveningStar` method:

INDICATORS

EveningStar() 1/1

```
EveningStar QuantConnect.Algorithm.CandlestickPatterns.EveningStar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `EveningStar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`EveningStar` - The new `EveningStar` indicator object.

Definition at line 372 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `EveningStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class EveningStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.eveningstar = EveningStar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.eveningstar.Update(bar)

        if self.eveningstar.IsReady:
            # The current value of self.eveningstar is represented by self.eveningstar.Current.Value
            self.Plot("EveningStar", "eveningstar", self.eveningstar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class EveningStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.eveningstar = EveningStar()
        self.RegisterIndicator(self.symbol, self.eveningstar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.eveningstar.IsReady:
            # The current value of self.eveningstar is represented by self.eveningstar.Current.Value
            self.Plot("EveningStar", "eveningstar", self.eveningstar.Current.Value)
```

The following reference table describes the EveningStar constructor:

INDICATORS

EveningStar() 1/3

```
EveningStar QuantConnect.Indicators.CandlestickPatterns.EveningStar (
string name,
*decimal penetration
)
```

Initializes a new instance of the EveningSta class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.
*decimal	penetration	*(Optional) (Optional)* Percentage of penetration of a candle within another candle. Default: 0.3m.

Return

EveningStar - The new EveningStar indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/EveningStar.cs.

INDICATORS

EveningStar() 2/3

```
EveningStar QuantConnect.Indicators.CandlestickPatterns.EveningStar (
decimal penetration
)
```

Initializes a new instance of the EveningSta class.

Show Details

Parameters		
decimal	penetration	Percentage of penetration of a candle within another candle.

Return

EveningStar - The new EveningStar indicator object.

Definition at line 67 of file Indicators/CandlestickPatterns/EveningStar.cs.

INDICATORS

EveningStar() 3/3

```
EveningStar QuantConnect.Indicators.CandlestickPatterns.EveningStar (
)
```

Initializes a new instance of the EveningSta class.

Show Details

This method requires no argument input.

Return

EveningStar - The new EveningStar indicator object.

Definition at line 75 of file Indicators/CandlestickPatterns/EveningStar.cs.

Supported Indicators

Gap Side By Side White

Introduction

Create a new Up/Down-gap side-by-side white lines candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using GapSideBySideWhite Indicator

To create an automatic indicators for `GapSideBySideWhite` , call the `GapSideBySideWhite` helper method from the `QCAlgorithm` class. The `GapSideBySideWhite` method creates a `GapSideBySideWhite` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class GapSideBySideWhiteAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.gapsidebysidewhite = self.GapSideBySideWhite(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.gapsidebysidewhite.IsReady:
            # The current value of self.gapsidebysidewhite is represented by
self.gapsidebysidewhite.Current.Value
            self.Plot("GapSideBySideWhite", "gapsidebysidewhite", self.gapsidebysidewhite.Current.Value)
```

The following reference table describes the `GapSideBySideWhite` method:

INDICATORS

GapSideBySideWhite() 1/1

```
GapSideBySideWhite QuantConnect.Algorithm.CandlestickPatterns.GapSideBySideWhite (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `GapSideBySideWhite` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`GapSideBySideWhite` - The new `GapSideBySideWhite` indicator object.

Definition at line 388 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `GapSideBySideWhite` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class GapSideBySideWhiteAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.gapsidebysidewhite = GapSideBySideWhite()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.gapsidebysidewhite.Update(bar)

        if self.gapsidebysidewhite.IsReady:
            # The current value of self.gapsidebysidewhite is represented by
self.gapsidebysidewhite.Current.Value
            self.Plot("GapSideBySideWhite", "gapsidebysidewhite", self.gapsidebysidewhite.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class GapSideBySideWhiteAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.gapsidebysidewhite = GapSideBySideWhite()
        self.RegisterIndicator(self.symbol, self.gapsidebysidewhite, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.gapsidebysidewhite.IsReady:
            # The current value of self.gapsidebysidewhite is represented by
self.gapsidebysidewhite.Current.Value
            self.Plot("GapSideBySideWhite", "gapsidebysidewhite", self.gapsidebysidewhite.Current.Value)
```

The following reference table describes the GapSideBySideWhite constructor:

INDICATORS

GapSideBySideWhite() 1/2

```
GapSideBySideWhite QuantConnect.Indicators.CandlestickPatterns.GapSideBySideWhite (
string name
)
```

Initializes a new instance of the GapSideBySideWhit class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

GapSideBySideWhite - The new GapSideBySideWhite indicator object.

Definition at line 48 of file Indicators/CandlestickPatterns/GapSideBySideWhite.cs.

INDICATORS

GapSideBySideWhite() 2/2

```
GapSideBySideWhite QuantConnect.Indicators.CandlestickPatterns.GapSideBySideWhite (
)
```

Initializes a new instance of the `GapSideBySideWhit` class.

Show Details

This method requires no argument input.

Return

`GapSideBySideWhite` - The new `GapSideBySideWhite` indicator object.

Definition at line 58 of file Indicators/CandlestickPatterns/GapSideBySideWhite.cs.

Supported Indicators

Gravestone Doji

Introduction

Create a new Gravestone Doji candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using GravestoneDoji Indicator

To create an automatic indicators for `GravestoneDoji` , call the `GravestoneDoji` helper method from the `QCAlgorithm` class. The `GravestoneDoji` method creates a `GravestoneDoji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class GravestoneDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.gravestonedoji = self.GravestoneDoji(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.gravestonedoji.IsReady:
            # The current value of self.gravestonedoji is represented by self.gravestonedoji.Current.Value
            self.Plot("GravestoneDoji", "gravestonedoji", self.gravestonedoji.Current.Value)
```

The following reference table describes the `GravestoneDoji` method:

INDICATORS

GravestoneDoji() 1/1

```
GravestoneDoji QuantConnect.Algorithm.CandlestickPatterns.GravestoneDoji (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `GravestoneDoji` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`GravestoneDoji` - The new `GravestoneDoji` indicator object.

Definition at line 404 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `GravestoneDoji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class GravestoneDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.gravestonedoji = GravestoneDoji()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.gravestonedoji.Update(bar)

        if self.gravestonedoji.IsReady:
            # The current value of self.gravestonedoji is represented by self.gravestonedoji.Current.Value
            self.Plot("GravestoneDoji", "gravestonedoji", self.gravestonedoji.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class GravestoneDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.gravestonedoji = GravestoneDoji()
        self.RegisterIndicator(self.symbol, self.gravestonedoji, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.gravestonedoji.IsReady:
            # The current value of self.gravestonedoji is represented by self.gravestonedoji.Current.Value
            self.Plot("GravestoneDoji", "gravestonedoji", self.gravestonedoji.Current.Value)
```

The following reference table describes the `GravestoneDoji` constructor:

INDICATORS

GravestoneDoji() 1/2

```
GravestoneDoji QuantConnect.Indicators.CandlestickPatterns.GravestoneDoji (
string name
)
```

Initializes a new instance of the `GravestoneDoj` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`GravestoneDoji` - The new `GravestoneDoji` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/GravestoneDoji.cs.

INDICATORS

GravestoneDoji() 2/2

```
GravestoneDoji QuantConnect.Indicators.CandlestickPatterns.GravestoneDoji (

)
```

Initializes a new instance of the `GravestoneDoj` class.

Show Details

This method requires no argument input.

Return

`GravestoneDoji` - The new `GravestoneDoji` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/GravestoneDoji.cs.

Supported Indicators

Hammer

Introduction

Create a new Hammer candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Hammer Indicator

To create an automatic indicators for `Hammer` , call the `Hammer` helper method from the `QCAlgorithm` class. The `Hammer` method creates a `Hammer` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class HammerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hammer = self.Hammer(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.hammer.IsReady:
            # The current value of self.hammer is represented by self.hammer.Current.Value
            self.Plot("Hammer", "hammer", self.hammer.Current.Value)
```

The following reference table describes the `Hammer` method:

INDICATORS

Hammer() 1/1

```
Hammer QuantConnect.Algorithm.CandlestickPatterns.Hammer (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Hammer` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Hammer` - The new `Hammer` indicator object.

Definition at line 420 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Hammer` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HammerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hammer = Hammer()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.hammer.Update(bar)

        if self.hammer.IsReady:
            # The current value of self.hammer is represented by self.hammer.Current.Value
            self.Plot("Hammer", "hammer", self.hammer.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class HammerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hammer = Hammer()
        self.RegisterIndicator(self.symbol, self.hammer, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.hammer.IsReady:
            # The current value of self.hammer is represented by self.hammer.Current.Value
            self.Plot("Hammer", "hammer", self.hammer.Current.Value)
```

The following reference table describes the `Hammer` constructor:

INDICATORS

Hammer() 1/2

```
Hammer QuantConnect.Indicators.CandlestickPatterns.Hammer (
string name
)
```

Initializes a new instance of the `Hamme` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Hammer` - The new `Hammer` indicator object.

Definition at line 50 of file Indicators/CandlestickPatterns/Hammer.cs.

INDICATORS

Hammer() 2/2

```
Hammer QuantConnect.Indicators.CandlestickPatterns.Hammer (

)
```

Initializes a new instance of the `Hamme` class.

Show Details

This method requires no argument input.

Return

`Hammer` - The new `Hammer` indicator object.

Definition at line 63 of file Indicators/CandlestickPatterns/Hammer.cs.

Supported Indicators

Hanging Man

Introduction

Create a new Hanging Man candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HangingMan Indicator

To create an automatic indicators for `HangingMan` , call the `HangingMan` helper method from the `QCAlgorithm` class. The `HangingMan` method creates a `HangingMan` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class HangingManAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hangingman = self.HangingMan(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.hangingman.IsReady:
            # The current value of self.hangingman is represented by self.hangingman.Current.Value
            self.Plot("HangingMan", "hangingman", self.hangingman.Current.Value)
```

The following reference table describes the `HangingMan` method:

INDICATORS

HangingMan() 1/1

```
HangingMan QuantConnect.Algorithm.CandlestickPatterns.HangingMan (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `HangingMan` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`HangingMan` - The new `HangingMan` indicator object.

Definition at line 436 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HangingMan` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HangingManAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hangingman = HangingMan()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.hangingman.Update(bar)

        if self.hangingman.IsReady:
            # The current value of self.hangingman is represented by self.hangingman.Current.Value
            self.Plot("HangingMan", "hangingman", self.hangingman.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class HangingManAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hangingman = HangingMan()
        self.RegisterIndicator(self.symbol, self.hangingman, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.hangingman.IsReady:
            # The current value of self.hangingman is represented by self.hangingman.Current.Value
            self.Plot("HangingMan", "hangingman", self.hangingman.Current.Value)
```

The following reference table describes the HangingMan constructor:

INDICATORS

HangingMan() 1/2

```
HangingMan QuantConnect.Indicators.CandlestickPatterns.HangingMan (
string name
)
```

Initializes a new instance of the HangingMa class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

HangingMan - The new HangingMan indicator object.

Definition at line 50 of file Indicators/CandlestickPatterns/HangingMan.cs.

INDICATORS

HangingMan() 2/2

```
HangingMan QuantConnect.Indicators.CandlestickPatterns.HangingMan (

)
```

Initializes a new instance of the `HangingMa` class.

Show Details

This method requires no argument input.

Return

`HangingMan` - The new `HangingMan` indicator object.

Definition at line 63 of file Indicators/CandlestickPatterns/HangingMan.cs.

Supported Indicators

Harami

Introduction

Create a new Harami candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Harami Indicator

To create an automatic indicators for `Harami` , call the `Harami` helper method from the `QCAlgorithm` class. The `Harami` method creates a `Harami` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class HaramiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.harami = self.Harami(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.harami.IsReady:
            # The current value of self.harami is represented by self.harami.Current.Value
            self.Plot("Harami", "harami", self.harami.Current.Value)
```

The following reference table describes the `Harami` method:

INDICATORS

Harami() 1/1

```
Harami QuantConnect.Algorithm.CandlestickPatterns.Harami (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Harami` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Harami` - The new `Harami` indicator object.

Definition at line 452 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Harami` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HaramiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.harami = Harami()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.harami.Update(bar)

        if self.harami.IsReady:
            # The current value of self.harami is represented by self.harami.Current.Value
            self.Plot("Harami", "harami", self.harami.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class HaramiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.harami = Harami()
        self.RegisterIndicator(self.symbol, self.harami, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.harami.IsReady:
            # The current value of self.harami is represented by self.harami.Current.Value
            self.Plot("Harami", "harami", self.harami.Current.Value)
```

The following reference table describes the `Harami` constructor:

INDICATORS

Harami() 1/2

```
Harami QuantConnect.Indicators.CandlestickPatterns.Harami (
string name
)
```

Initializes a new instance of the `Haram` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`Harami` - The new `Harami` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/Harami.cs.

INDICATORS

Harami() 2/2

```
Harami QuantConnect.Indicators.CandlestickPatterns.Harami (

)
```

Initializes a new instance of the `Haram` class.

Show Details

This method requires no argument input.

Return

`Harami` - The new `Harami` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/Harami.cs.

Supported Indicators

Harami Cross

Introduction

Create a new Harami Cross candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HaramiCross Indicator

To create an automatic indicators for `HaramiCross` , call the `HaramiCross` helper method from the `QCAlgorithm` class. The `HaramiCross` method creates a `HaramiCross` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class HaramiCrossAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.haramicross = self.HaramiCross(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.haramicross.IsReady:
            # The current value of self.haramicross is represented by self.haramicross.Current.Value
            self.Plot("HaramiCross", "haramicross", self.haramicross.Current.Value)
```

The following reference table describes the `HaramiCross` method:

INDICATORS

HaramiCross() 1/1

```
HaramiCross QuantConnect.Algorithm.CandlestickPatterns.HaramiCross (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `HaramiCross` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`HaramiCross` - The new `HaramiCross` indicator object.

Definition at line 468 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HaramiCross` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HaramiCrossAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.haramicross = HaramiCross()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.haramicross.Update(bar)

        if self.haramicross.IsReady:
            # The current value of self.haramicross is represented by self.haramicross.Current.Value
            self.Plot("HaramiCross", "haramicross", self.haramicross.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class HaramiCrossAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.haramicross = HaramiCross()
        self.RegisterIndicator(self.symbol, self.haramicross, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.haramicross.IsReady:
            # The current value of self.haramicross is represented by self.haramicross.Current.Value
            self.Plot("HaramiCross", "haramicross", self.haramicross.Current.Value)
```

The following reference table describes the `HaramiCross` constructor:

INDICATORS

HaramiCross() 1/2

```
HaramiCross QuantConnect.Indicators.CandlestickPatterns.HaramiCross (
string name
)
```

Initializes a new instance of the `HaramiCros` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`HaramiCross` - The new `HaramiCross` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/HaramiCross.cs.

INDICATORS

HaramiCross() 2/2

```
HaramiCross QuantConnect.Indicators.CandlestickPatterns.HaramiCross (
)
```

Initializes a new instance of the `HaramiCros` class.

Show Details

This method requires no argument input.

Return

`HaramiCross` - The new `HaramiCross` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/HaramiCross.cs.

Supported Indicators

High Wave Candle

Introduction

Create a new High-Wave Candle candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HighWaveCandle Indicator

To create an automatic indicators for `HighWaveCandle` , call the `HighWaveCandle` helper method from the `QCAlgorithm` class. The `HighWaveCandle` method creates a `HighWaveCandle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class HighWaveCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.highwavecandle = self.HighWaveCandle(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.highwavecandle.IsReady:
            # The current value of self.highwavecandle is represented by self.highwavecandle.Current.Value
            self.Plot("HighWaveCandle", "highwavecandle", self.highwavecandle.Current.Value)
```

The following reference table describes the `HighWaveCandle` method:

INDICATORS

HighWaveCandle() 1/1

```
HighWaveCandle QuantConnect.Algorithm.CandlestickPatterns.HighWaveCandle (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `HighWaveCandle` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`HighWaveCandle` - The new `HighWaveCandle` indicator object.

Definition at line 484 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HighWaveCandle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HighWaveCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.highwavecandle = HighWaveCandle()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.highwavecandle.Update(bar)

        if self.highwavecandle.IsReady:
            # The current value of self.highwavecandle is represented by self.highwavecandle.Current.Value
            self.Plot("HighWaveCandle", "highwavecandle", self.highwavecandle.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class HighWaveCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.highwavecandle = HighWaveCandle()
        self.RegisterIndicator(self.symbol, self.highwavecandle, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.highwavecandle.IsReady:
            # The current value of self.highwavecandle is represented by self.highwavecandle.Current.Value
            self.Plot("HighWaveCandle", "highwavecandle", self.highwavecandle.Current.Value)
```

The following reference table describes the `HighWaveCandle` constructor:

INDICATORS

HighWaveCandle() 1/2

```
HighWaveCandle QuantConnect.Indicators.CandlestickPatterns.HighWaveCandle (
string name
)
```

Initializes a new instance of the `HighWaveCandl` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`HighWaveCandle` - The new `HighWaveCandle` indicator object.

Definition at line 44 of file Indicators/CandlestickPatterns/HighWaveCandle.cs.

INDICATORS

HighWaveCandle() 2/2

```
HighWaveCandle QuantConnect.Indicators.CandlestickPatterns.HighWaveCandle (
)
```

Initializes a new instance of the `HighWaveCandl` class.

Show Details

This method requires no argument input.

Return

`HighWaveCandle` - The new `HighWaveCandle` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/HighWaveCandle.cs.

Supported Indicators

Hikkake

Introduction

Create a new Hikkake candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Hikkake Indicator

To create an automatic indicators for `Hikkake` , call the `Hikkake` helper method from the `QCAlgorithm` class. The `Hikkake` method creates a `Hikkake` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class HikkakeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hikkake = self.Hikkake(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.hikkake.IsReady:
            # The current value of self.hikkake is represented by self.hikkake.Current.Value
            self.Plot("Hikkake", "hikkake", self.hikkake.Current.Value)
```

The following reference table describes the `Hikkake` method:

INDICATORS

Hikkake() 1/1

```
Hikkake QuantConnect.Algorithm.CandlestickPatterns.Hikkake (
 Symbol symbol,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
 )
```

Creates a new `Hikkake` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Hikkake` - The new `Hikkake` indicator object.

Definition at line 500 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Hikkake` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HikkakeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hikkake = Hikkake()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.hikkake.Update(bar)

        if self.hikkake.IsReady:
            # The current value of self.hikkake is represented by self.hikkake.Current.Value
            self.Plot("Hikkake", "hikkake", self.hikkake.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class HikkakeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hikkake = Hikkake()
        self.RegisterIndicator(self.symbol, self.hikkake, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.hikkake.IsReady:
            # The current value of self.hikkake is represented by self.hikkake.Current.Value
            self.Plot("Hikkake", "hikkake", self.hikkake.Current.Value)
```

The following reference table describes the `Hikkake` constructor:

INDICATORS

Hikkake() 1/2

```
Hikkake QuantConnect.Indicators.CandlestickPatterns.Hikkake (
string name
)
```

Initializes a new instance of the `Hikkak` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Hikkake` - The new `Hikkake` indicator object.

Definition at line 44 of file Indicators/CandlestickPatterns/Hikkake.cs.

INDICATORS

Hikkake() 2/2

```
Hikkake QuantConnect.Indicators.CandlestickPatterns.Hikkake (

)
```

Initializes a new instance of the `Hikkak` class.

Show Details

This method requires no argument input.

Return

`Hikkake` - The new `Hikkake` indicator object.

Definition at line 52 of file Indicators/CandlestickPatterns/Hikkake.cs.

Supported Indicators

Hikkake Modified

Introduction

Create a new Hikkake Modified candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HikkakeModified Indicator

To create an automatic indicators for `HikkakeModified` , call the `HikkakeModified` helper method from the `QCAlgorithm` class. The `HikkakeModified` method creates a `HikkakeModified` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class HikkakeModifiedAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hikkakemodified = self.HikkakeModified(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.hikkakemodified.IsReady:
            # The current value of self.hikkakemodified is represented by
self.hikkakemodified.Current.Value
            self.Plot("HikkakeModified", "hikkakemodified", self.hikkakemodified.Current.Value)
```

PY

The following reference table describes the `HikkakeModified` method:

INDICATORS

HikkakeModified() 1/1

```
HikkakeModified QuantConnect.Algorithm.CandlestickPatterns.HikkakeModified (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `HikkakeModified` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`HikkakeModified` - The new `HikkakeModified` indicator object.

Definition at line 516 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HikkakeModified` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```
class HikkakeModifiedAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hikkakemodified = HikkakeModified()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.hikkakemodified.Update(bar)

        if self.hikkakemodified.IsReady:
            # The current value of self.hikkakemodified is represented by
self.hikkakemodified.Current.Value
            self.Plot("HikkakeModified", "hikkakemodified", self.hikkakemodified.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class HikkakeModifiedAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hikkakemodified = HikkakeModified()
        self.RegisterIndicator(self.symbol, self.hikkakemodified, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.hikkakemodified.IsReady:
            # The current value of self.hikkakemodified is represented by
self.hikkakemodified.Current.Value
            self.Plot("HikkakeModified", "hikkakemodified", self.hikkakemodified.Current.Value)
```

The following reference table describes the `HikkakeModified` constructor:

INDICATORS

HikkakeModified() 1/2

```
HikkakeModified QuantConnect.Indicators.CandlestickPatterns.HikkakeModified (
string name
)
```

Initializes a new instance of the `HikkakeModifie` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`HikkakeModified` - The new `HikkakeModified` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/HikkakeModified.cs.

INDICATORS

HikkakeModified() 2/2

```
HikkakeModified QuantConnect.Indicators.CandlestickPatterns.HikkakeModified (
)
```

Initializes a new instance of the `HikkakeModifie` class.

Show Details

This method requires no argument input.

Return

`HikkakeModified` - The new `HikkakeModified` indicator object.

Definition at line 62 of file Indicators/CandlestickPatterns/HikkakeModified.cs.

Supported Indicators

Homing Pigeon

Introduction

Create a new Homing Pigeon candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HomingPigeon Indicator

To create an automatic indicators for `HomingPigeon` , call the `HomingPigeon` helper method from the `QCAlgorithm` class. The `HomingPigeon` method creates a `HomingPigeon` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class HomingPigeonAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.homingpigeon = self.HomingPigeon(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.homingpigeon.IsReady:
            # The current value of self.homingpigeon is represented by self.homingpigeon.Current.Value
            self.Plot("HomingPigeon", "homingpigeon", self.homingpigeon.Current.Value)
```

The following reference table describes the `HomingPigeon` method:

INDICATORS

HomingPigeon() 1/1

```
HomingPigeon QuantConnect.Algorithm.CandlestickPatterns.HomingPigeon (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `HomingPigeon` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`HomingPigeon` - The new `HomingPigeon` indicator object.

Definition at line 532 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HomingPigeon` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class HomingPigeonAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.homingpigeon = HomingPigeon()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.homingpigeon.Update(bar)

        if self.homingpigeon.IsReady:
            # The current value of self.homingpigeon is represented by self.homingpigeon.Current.Value
            self.Plot("HomingPigeon", "homingpigeon", self.homingpigeon.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class HomingPigeonAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.homingpigeon = HomingPigeon()
        self.RegisterIndicator(self.symbol, self.homingpigeon, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.homingpigeon.IsReady:
            # The current value of self.homingpigeon is represented by self.homingpigeon.Current.Value
            self.Plot("HomingPigeon", "homingpigeon", self.homingpigeon.Current.Value)
```

The following reference table describes the `HomingPigeon` constructor:

INDICATORS

HomingPigeon() 1/2

```
HomingPigeon QuantConnect.Indicators.CandlestickPatterns.HomingPigeon (
string name
)
```

Initializes a new instance of the `HomingPigeo` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`HomingPigeon` - The new `HomingPigeon` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/HomingPigeon.cs.

INDICATORS

HomingPigeon() 2/2

```
HomingPigeon QuantConnect.Indicators.CandlestickPatterns.HomingPigeon (

)
```

Initializes a new instance of the `HomingPigeo` class.

Show Details

This method requires no argument input.

Return

`HomingPigeon` - The new `HomingPigeon` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/HomingPigeon.cs.

Supported Indicators

Identical Three Crows

Introduction

Create a new Identical Three Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using IdenticalThreeCrows Indicator

To create an automatic indicators for `IdenticalThreeCrows` , call the `IdenticalThreeCrows` helper method from the `QCAlgorithm` class. The `IdenticalThreeCrows` method creates a `IdenticalThreeCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class IdenticalThreeCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.identicalthreecrows = self.IdenticalThreeCrows(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.identicalthreecrows.IsReady:
            # The current value of self.identicalthreecrows is represented by
self.identicalthreecrows.Current.Value
            self.Plot("IdenticalThreeCrows", "identicalthreecrows",
self.identicalthreecrows.Current.Value)
```

The following reference table describes the `IdenticalThreeCrows` method:

INDICATORS

IdenticalThreeCrows() 1/1

```
IdenticalThreeCrows QuantConnect.Algorithm.CandlestickPatterns.IdenticalThreeCrows (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `IdenticalThreeCrows` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`IdenticalThreeCrows` - The new `IdenticalThreeCrows` indicator object.

Definition at line 548 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `IdenticalThreeCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class IdenticalThreeCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.identicalthreecrows = IdenticalThreeCrows()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.identicalthreecrows.Update(bar)

        if self.identicalthreecrows.IsReady:
            # The current value of self.identicalthreecrows is represented by
self.identicalthreecrows.Current.Value
            self.Plot("IdenticalThreeCrows", "identicalthreecrows",
self.identicalthreecrows.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class IdenticalThreeCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.identicalthreecrows = IdenticalThreeCrows()
        self.RegisterIndicator(self.symbol, self.identicalthreecrows, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.identicalthreecrows.IsReady:
            # The current value of self.identicalthreecrows is represented by
self.identicalthreecrows.Current.Value
            self.Plot("IdenticalThreeCrows", "identicalthreecrows",
self.identicalthreecrows.Current.Value)
```

The following reference table describes the `IdenticalThreeCrows` constructor:

INDICATORS

IdenticalThreeCrows() 1/2

```
IdenticalThreeCrows QuantConnect.Indicators.CandlestickPatterns.IdenticalThreeCrows (
string name
)
```

Initializes a new instance of the `IdenticalThreeCrow` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`IdenticalThreeCrows` - The new `IdenticalThreeCrows` indicator object.

Definition at line 48 of file Indicators/CandlestickPatterns/IdenticalThreeCrows.cs.

INDICATORS

IdenticalThreeCrows() 2/2

```
IdenticalThreeCrows QuantConnect.Indicators.CandlestickPatterns.IdenticalThreeCrows (

)
```

Initializes a new instance of the `IdenticalThreeCrow` class.

Show Details

This method requires no argument input.

Return

`IdenticalThreeCrows` - The new `IdenticalThreeCrows` indicator object.

Definition at line 58 of file Indicators/CandlestickPatterns/IdenticalThreeCrows.cs.

Supported Indicators

In Neck

Introduction

Create a new In-Neck candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using InNeck Indicator

To create an automatic indicators for `InNeck` , call the `InNeck` helper method from the `QCAlgorithm` class. The `InNeck` method creates a `InNeck` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class InNeckAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.inneck = self.InNeck(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.inneck.IsReady:
            # The current value of self.inneck is represented by self.inneck.Current.Value
            self.Plot("InNeck", "inneck", self.inneck.Current.Value)
```

The following reference table describes the `InNeck` method:

INDICATORS

InNeck() 1/1

```
InNeck QuantConnect.Algorithm.CandlestickPatterns.InNeck (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `InNeck` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`InNeck` - The new `InNeck` indicator object.

Definition at line 564 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `InNeck` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class InNeckAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.inneck = InNeck()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.inneck.Update(bar)

        if self.inneck.IsReady:
            # The current value of self.inneck is represented by self.inneck.Current.Value
            self.Plot("InNeck", "inneck", self.inneck.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class InNeckAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.inneck = InNeck()
        self.RegisterIndicator(self.symbol, self.inneck, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.inneck.IsReady:
            # The current value of self.inneck is represented by self.inneck.Current.Value
            self.Plot("InNeck", "inneck", self.inneck.Current.Value)
```

The following reference table describes the `InNeck` constructor:

INDICATORS

InNeck() 1/2

```
InNeck QuantConnect.Indicators.CandlestickPatterns.InNeck (
string name
)
```

Initializes a new instance of the `InNec` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`InNeck` - The new `InNeck` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/InNeck.cs.

INDICATORS

InNeck() 2/2

```
InNeck QuantConnect.Indicators.CandlestickPatterns.InNeck (

)
```

Initializes a new instance of the `InNec` class.

Show Details

This method requires no argument input.

Return

`InNeck` - The new `InNeck` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/InNeck.cs.

Supported Indicators

Inverted Hammer

Introduction

Create a new Inverted Hammer candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using InvertedHammer Indicator

To create an automatic indicators for `InvertedHammer` , call the `InvertedHammer` helper method from the `QCAlgorithm` class. The `InvertedHammer` method creates a `InvertedHammer` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class InvertedHammerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.invertedhammer = self.InvertedHammer(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.invertedhammer.IsReady:
            # The current value of self.invertedhammer is represented by self.invertedhammer.Current.Value
            self.Plot("InvertedHammer", "invertedhammer", self.invertedhammer.Current.Value)
```

The following reference table describes the `InvertedHammer` method:

INDICATORS

InvertedHammer() 1/1

```
InvertedHammer QuantConnect.Algorithm.CandlestickPatterns.InvertedHammer (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `InvertedHammer` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`InvertedHammer` - The new `InvertedHammer` indicator object.

Definition at line 580 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `InvertedHammer` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class InvertedHammerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.invertedhammer = InvertedHammer()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.invertedhammer.Update(bar)

        if self.invertedhammer.IsReady:
            # The current value of self.invertedhammer is represented by self.invertedhammer.Current.Value
            self.Plot("InvertedHammer", "invertedhammer", self.invertedhammer.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class InvertedHammerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.invertedhammer = InvertedHammer()
        self.RegisterIndicator(self.symbol, self.invertedhammer, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.invertedhammer.IsReady:
            # The current value of self.invertedhammer is represented by self.invertedhammer.Current.Value
            self.Plot("InvertedHammer", "invertedhammer", self.invertedhammer.Current.Value)
```

The following reference table describes the `InvertedHammer` constructor:

INDICATORS

InvertedHammer() 1/2

```
InvertedHammer QuantConnect.Indicators.CandlestickPatterns.InvertedHammer (
string name
)
```

Initializes a new instance of the `InvertedHamme` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`InvertedHammer` - The new `InvertedHammer` indicator object.

Definition at line 48 of file Indicators/CandlestickPatterns/InvertedHammer.cs.

INDICATORS

InvertedHammer() 2/2

```
InvertedHammer QuantConnect.Indicators.CandlestickPatterns.InvertedHammer (

)
```

Initializes a new instance of the `InvertedHamme` class.

Show Details

This method requires no argument input.

Return

`InvertedHammer` - The new `InvertedHammer` indicator object.

Definition at line 60 of file Indicators/CandlestickPatterns/InvertedHammer.cs.

Supported Indicators

Kicking

Introduction

Create a new Kicking candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Kicking Indicator

To create an automatic indicators for `Kicking` , call the `Kicking` helper method from the `QCAlgorithm` class. The `Kicking` method creates a `Kicking` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class KickingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kicking = self.Kicking(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.kicking.IsReady:
            # The current value of self.kicking is represented by self.kicking.Current.Value
            self.Plot("Kicking", "kicking", self.kicking.Current.Value)
```

The following reference table describes the `Kicking` method:

INDICATORS

Kicking() 1/1

```
Kicking QuantConnect.Algorithm.CandlestickPatterns.Kicking (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Kicking` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Kicking` - The new `Kicking` indicator object.

Definition at line 596 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Kicking` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class KickingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kicking = Kicking()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.kicking.Update(bar)

        if self.kicking.IsReady:
            # The current value of self.kicking is represented by self.kicking.Current.Value
            self.Plot("Kicking", "kicking", self.kicking.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class KickingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kicking = Kicking()
        self.RegisterIndicator(self.symbol, self.kicking, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.kicking.IsReady:
            # The current value of self.kicking is represented by self.kicking.Current.Value
            self.Plot("Kicking", "kicking", self.kicking.Current.Value)
```

The following reference table describes the `Kicking` constructor:

INDICATORS

Kicking() 1/2

```
Kicking QuantConnect.Indicators.CandlestickPatterns.Kicking (
string name
)
```

Initializes a new instance of the `Kickin` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Kicking` - The new `Kicking` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/Kicking.cs.

INDICATORS

Kicking() 2/2

```
Kicking QuantConnect.Indicators.CandlestickPatterns.Kicking (

)
```

Initializes a new instance of the `Kickin` class.

Show Details

This method requires no argument input.

Return

`Kicking` - The new `Kicking` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/Kicking.cs.

Supported Indicators

Kicking By Length

Introduction

Create a new Kicking (bull/bear determined by the longer marubozu) candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using KickingByLength Indicator

To create an automatic indicators for `KickingByLength` , call the `KickingByLength` helper method from the `QCAlgorithm` class. The `KickingByLength` method creates a `KickingByLength` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class KickingByLengthAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kickingbylength = self.KickingByLength(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.kickingbylength.IsReady:
            # The current value of self.kickingbylength is represented by
self.kickingbylength.Current.Value
            self.Plot("KickingByLength", "kickingbylength", self.kickingbylength.Current.Value)
```

The following reference table describes the `KickingByLength` method:

INDICATORS

KickingByLength() 1/1

```
KickingByLength QuantConnect.Algorithm.CandlestickPatterns.KickingByLength (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `KickingByLength` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`KickingByLength` - The new `KickingByLength` indicator object.

Definition at line 612 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `KickingByLength` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class KickingByLengthAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kickingbylength = KickingByLength()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.kickingbylength.Update(bar)

        if self.kickingbylength.IsReady:
            # The current value of self.kickingbylength is represented by
self.kickingbylength.Current.Value
            self.Plot("KickingByLength", "kickingbylength", self.kickingbylength.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class KickingByLengthAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kickingbylength = KickingByLength()
        self.RegisterIndicator(self.symbol, self.kickingbylength, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.kickingbylength.IsReady:
            # The current value of self.kickingbylength is represented by
self.kickingbylength.Current.Value
            self.Plot("KickingByLength", "kickingbylength", self.kickingbylength.Current.Value)
```

The following reference table describes the `KickingByLength` constructor:

INDICATORS

KickingByLength() 1/2

```
KickingByLength QuantConnect.Indicators.CandlestickPatterns.KickingByLength (
string name
)
```

Initializes a new instance of the `KickingByLengt` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`KickingByLength` - The new `KickingByLength` indicator object.

Definition at line 46 of file Indicators/CandlestickPatterns/KickingByLength.cs.

INDICATORS

KickingByLength() 2/2

```
KickingByLength QuantConnect.Indicators.CandlestickPatterns.KickingByLength (
)
```

Initializes a new instance of the `KickingByLengt` class.

Show Details

This method requires no argument input.

Return

`KickingByLength` - The new `KickingByLength` indicator object.

Definition at line 56 of file Indicators/CandlestickPatterns/KickingByLength.cs.

Supported Indicators

Ladder Bottom

Introduction

Create a new Ladder Bottom candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using LadderBottom Indicator

To create an automatic indicators for `LadderBottom` , call the `LadderBottom` helper method from the `QCAlgorithm` class. The `LadderBottom` method creates a `LadderBottom` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class LadderBottomAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ladderbottom = self.LadderBottom(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.ladderbottom.IsReady:
            # The current value of self.ladderbottom is represented by self.ladderbottom.Current.Value
            self.Plot("LadderBottom", "ladderbottom", self.ladderbottom.Current.Value)
```

The following reference table describes the `LadderBottom` method:

INDICATORS

LadderBottom() 1/1

```
LadderBottom QuantConnect.Algorithm.CandlestickPatterns.LadderBottom (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `LadderBottom` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`LadderBottom` - The new `LadderBottom` indicator object.

Definition at line 628 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `LadderBottom` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class LadderBottomAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ladderbottom = LadderBottom()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ladderbottom.Update(bar)

        if self.ladderbottom.IsReady:
            # The current value of self.ladderbottom is represented by self.ladderbottom.Current.Value
            self.Plot("LadderBottom", "ladderbottom", self.ladderbottom.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class LadderBottomAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ladderbottom = LadderBottom()
        self.RegisterIndicator(self.symbol, self.ladderbottom, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ladderbottom.IsReady:
            # The current value of self.ladderbottom is represented by self.ladderbottom.Current.Value
            self.Plot("LadderBottom", "ladderbottom", self.ladderbottom.Current.Value)
```

The following reference table describes the `LadderBottom` constructor:

INDICATORS

LadderBottom() 1/2

```
LadderBottom QuantConnect.Indicators.CandlestickPatterns.LadderBottom (
string name
)
```

Initializes a new instance of the `LadderBotto` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`LadderBottom` - The new `LadderBottom` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/LadderBottom.cs.

INDICATORS

LadderBottom() 2/2

```
LadderBottom QuantConnect.Indicators.CandlestickPatterns.LadderBottom (

)
```

Initializes a new instance of the `LadderBotto` class.

Show Details

This method requires no argument input.

Return

`LadderBottom` - The new `LadderBottom` indicator object.

Definition at line 52 of file Indicators/CandlestickPatterns/LadderBottom.cs.

Supported Indicators

Long Legged Doji

Introduction

Create a new Long Legged Doji candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using LongLeggedDoji Indicator

To create an automatic indicators for `LongLeggedDoji` , call the `LongLeggedDoji` helper method from the `QCAlgorithm` class. The `LongLeggedDoji` method creates a `LongLeggedDoji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class LongLeggedDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.longleggeddoji = self.LongLeggedDoji(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.longleggeddoji.IsReady:
            # The current value of self.longleggeddoji is represented by self.longleggeddoji.Current.Value
            self.Plot("LongLeggedDoji", "longleggeddoji", self.longleggeddoji.Current.Value)
```

The following reference table describes the `LongLeggedDoji` method:

INDICATORS

LongLeggedDoji() 1/1

```
LongLeggedDoji QuantConnect.Algorithm.CandlestickPatterns.LongLeggedDoji (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `LongLeggedDoji` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`LongLeggedDoji` - The new `LongLeggedDoji` indicator object.

Definition at line 644 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `LongLeggedDoji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class LongLeggedDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.longleggeddoji = LongLeggedDoji()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.longleggeddoji.Update(bar)

        if self.longleggeddoji.IsReady:
            # The current value of self.longleggeddoji is represented by self.longleggeddoji.Current.Value
            self.Plot("LongLeggedDoji", "longleggeddoji", self.longleggeddoji.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class LongLeggedDojiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.longleggeddoji = LongLeggedDoji()
        self.RegisterIndicator(self.symbol, self.longleggeddoji, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.longleggeddoji.IsReady:
            # The current value of self.longleggeddoji is represented by self.longleggeddoji.Current.Value
            self.Plot("LongLeggedDoji", "longleggeddoji", self.longleggeddoji.Current.Value)
```

The following reference table describes the `LongLeggedDoji` constructor:

INDICATORS

LongLeggedDoji() 1/2

```
LongLeggedDoji QuantConnect.Indicators.CandlestickPatterns.LongLeggedDoji (
string name
)
```

Initializes a new instance of the `LongLeggedDoj` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`LongLeggedDoji` - The new `LongLeggedDoji` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/LongLeggedDoji.cs.

INDICATORS

LongLeggedDoji() 2/2

```
LongLeggedDoji QuantConnect.Indicators.CandlestickPatterns.LongLeggedDoji (
)
```

Initializes a new instance of the `LongLeggedDoj` class.

Show Details

This method requires no argument input.

Return

`LongLeggedDoji` - The new `LongLeggedDoji` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/LongLeggedDoji.cs.

Supported Indicators

Long Line Candle

Introduction

Create a new Long Line Candle candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using LongLineCandle Indicator

To create an automatic indicators for `LongLineCandle` , call the `LongLineCandle` helper method from the `QCAlgorithm` class. The `LongLineCandle` method creates a `LongLineCandle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class LongLineCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.longlinecandle = self.LongLineCandle(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.longlinecandle.IsReady:
            # The current value of self.longlinecandle is represented by self.longlinecandle.Current.Value
            self.Plot("LongLineCandle", "longlinecandle", self.longlinecandle.Current.Value)
```

The following reference table describes the `LongLineCandle` method:

INDICATORS

LongLineCandle() 1/1

```
LongLineCandle QuantConnect.Algorithm.CandlestickPatterns.LongLineCandle (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `LongLineCandle` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`LongLineCandle` - The new `LongLineCandle` indicator object.

Definition at line 660 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `LongLineCandle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class LongLineCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.longlinecandle = LongLineCandle()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.longlinecandle.Update(bar)

        if self.longlinecandle.IsReady:
            # The current value of self.longlinecandle is represented by self.longlinecandle.Current.Value
            self.Plot("LongLineCandle", "longlinecandle", self.longlinecandle.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class LongLineCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.longlinecandle = LongLineCandle()
        self.RegisterIndicator(self.symbol, self.longlinecandle, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.longlinecandle.IsReady:
            # The current value of self.longlinecandle is represented by self.longlinecandle.Current.Value
            self.Plot("LongLineCandle", "longlinecandle", self.longlinecandle.Current.Value)
```

The following reference table describes the `LongLineCandle` constructor:

INDICATORS

LongLineCandle() 1/2

```
LongLineCandle QuantConnect.Indicators.CandlestickPatterns.LongLineCandle (
string name
)
```

Initializes a new instance of the `LongLineCandl` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`LongLineCandle` - The new `LongLineCandle` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/LongLineCandle.cs.

INDICATORS

LongLineCandle() 2/2

```
LongLineCandle QuantConnect.Indicators.CandlestickPatterns.LongLineCandle (
)
```

Initializes a new instance of the `LongLineCandl` class.

Show Details

This method requires no argument input.

Return

`LongLineCandle` - The new `LongLineCandle` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/LongLineCandle.cs.

Supported Indicators

Marubozu

Introduction

Create a new Marubozu candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Marubozu Indicator

To create an automatic indicators for `Marubozu` , call the `Marubozu` helper method from the `QCAlgorithm` class. The `Marubozu` method creates a `Marubozu` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MarubozuAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.marubozu = self.Marubozu(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.marubozu.IsReady:
            # The current value of self.marubozu is represented by self.marubozu.Current.Value
            self.Plot("Marubozu", "marubozu", self.marubozu.Current.Value)
```

The following reference table describes the `Marubozu` method:

INDICATORS

Marubozu() 1/1

```
Marubozu QuantConnect.Algorithm.CandlestickPatterns.Marubozu (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Marubozu` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Marubozu` - The new `Marubozu` indicator object.

Definition at line 676 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Marubozu` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MarubozuAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.marubozu = Marubozu()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.marubozu.Update(bar)

        if self.marubozu.IsReady:
            # The current value of self.marubozu is represented by self.marubozu.Current.Value
            self.Plot("Marubozu", "marubozu", self.marubozu.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class MarubozuAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.marubozu = Marubozu()
        self.RegisterIndicator(self.symbol, self.marubozu, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.marubozu.IsReady:
            # The current value of self.marubozu is represented by self.marubozu.Current.Value
            self.Plot("Marubozu", "marubozu", self.marubozu.Current.Value)
```

The following reference table describes the `Marubozu` constructor:

INDICATORS

Marubozu() 1/2

```
Marubozu QuantConnect.Indicators.CandlestickPatterns.Marubozu (
string name
)
```

Initializes a new instance of the `Maruboz` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`Marubozu` - The new `Marubozu` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/Marubozu.cs.

INDICATORS

Marubozu() 2/2

```
Marubozu QuantConnect.Indicators.CandlestickPatterns.Marubozu (

)
```

Initializes a new instance of the `Maruboz` class.

Show Details

This method requires no argument input.

Return

`Marubozu` - The new `Marubozu` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/Marubozu.cs.

Supported Indicators

Mat Hold

Introduction

Create a new Mat Hold candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MatHold Indicator

To create an automatic indicators for `MatHold` , call the `MatHold` helper method from the `QCAlgorithm` class. The `MatHold` method creates a `MatHold` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MatHoldAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mathold = self.MatHold(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.mathold.IsReady:
            # The current value of self.mathold is represented by self.mathold.Current.Value
            self.Plot("MatHold", "mathold", self.mathold.Current.Value)
```

The following reference table describes the `MatHold` method:

INDICATORS

MathHold() 1/1

```
MatHold QuantConnect.Algorithm.CandlestickPatterns.MatHold (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `MatHold` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`MatHold` - The new `MatHold` indicator object.

Definition at line 709 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MatHold` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MatHoldAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mathold = MatHold()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.mathold.Update(bar)

        if self.mathold.IsReady:
            # The current value of self.mathold is represented by self.mathold.Current.Value
            self.Plot("MatHold", "mathold", self.mathold.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MatHoldAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mathold = MatHold()
        self.RegisterIndicator(self.symbol, self.mathold, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.mathold.IsReady:
            # The current value of self.mathold is represented by self.mathold.Current.Value
            self.Plot("MatHold", "mathold", self.mathold.Current.Value)
```

The following reference table describes the `MatHold` constructor:

INDICATORS

MatHold() 1/3

```
MatHold QuantConnect.Indicators.CandlestickPatterns.MatHold (
string name,
*decimal penetration
)
```

Initializes a new instance of the `MatHol` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	penetration	*(Optional) (Optional)* Percentage of penetration of a candle within another candle. Default: 0.5m.

Return

`MatHold` - The new `MatHold` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/MatHold.cs.

INDICATORS

MatHold() 2/3

```
MatHold QuantConnect.Indicators.CandlestickPatterns.MatHold (
decimal penetration
)
```

Initializes a new instance of the `MatHol` class.

Show Details

Parameters		
`decimal`	penetration	Percentage of penetration of a candle within another candle.

Return

`MatHold` - The new `MatHold` indicator object.

Definition at line 67 of file Indicators/CandlestickPatterns/MatHold.cs.

INDICATORS

MatHold() 3/3

```
MatHold QuantConnect.Indicators.CandlestickPatterns.MatHold (
)
```

Initializes a new instance of the `MatHol` class.

Show Details

This method requires no argument input.

Return

`MatHold` - The new `MatHold` indicator object.

Definition at line 75 of file Indicators/CandlestickPatterns/MatHold.cs.

Supported Indicators

Matching Low

Introduction

Create a new Matching Low candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MatchingLow Indicator

To create an automatic indicators for `MatchingLow` , call the `MatchingLow` helper method from the `QCAlgorithm` class. The `MatchingLow` method creates a `MatchingLow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MatchingLowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.matchinglow = self.MatchingLow(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.matchinglow.IsReady:
            # The current value of self.matchinglow is represented by self.matchinglow.Current.Value
            self.Plot("MatchingLow", "matchinglow", self.matchinglow.Current.Value)
```

The following reference table describes the `MatchingLow` method:

INDICATORS

MatchingLow() 1/1

```
MatchingLow QuantConnect.Algorithm.CandlestickPatterns.MatchingLow (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `MatchingLow` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`MatchingLow` - The new `MatchingLow` indicator object.

Definition at line 692 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MatchingLow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MatchingLowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.matchinglow = MatchingLow()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.matchinglow.Update(bar)

        if self.matchinglow.IsReady:
            # The current value of self.matchinglow is represented by self.matchinglow.Current.Value
            self.Plot("MatchingLow", "matchinglow", self.matchinglow.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class MatchingLowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.matchinglow = MatchingLow()
        self.RegisterIndicator(self.symbol, self.matchinglow, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.matchinglow.IsReady:
            # The current value of self.matchinglow is represented by self.matchinglow.Current.Value
            self.Plot("MatchingLow", "matchinglow", self.matchinglow.Current.Value)
```

The following reference table describes the `MatchingLow` constructor:

INDICATORS

MatchingLow() 1/2

```
MatchingLow QuantConnect.Indicators.CandlestickPatterns.MatchingLow (
string name
)
```

Initializes a new instance of the `MatchingLo` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`MatchingLow` - The new `MatchingLow` indicator object.

Definition at line 40 of file Indicators/CandlestickPatterns/MatchingLow.cs.

INDICATORS

MatchingLow() 2/2

```
MatchingLow QuantConnect.Indicators.CandlestickPatterns.MatchingLow (

)
```

Initializes a new instance of the `MatchingLo` class.

Show Details

This method requires no argument input.

Return

`MatchingLow` - The new `MatchingLow` indicator object.

Definition at line 49 of file Indicators/CandlestickPatterns/MatchingLow.cs.

Supported Indicators

Morning Doji Star

Introduction

Create a new Morning Doji Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MorningDojiStar Indicator

To create an automatic indicators for `MorningDojiStar` , call the `MorningDojiStar` helper method from the `QCAlgorithm` class. The `MorningDojiStar` method creates a `MorningDojiStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MorningDojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.morningdojistar = self.MorningDojiStar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.morningdojistar.IsReady:
            # The current value of self.morningdojistar is represented by
self.morningdojistar.Current.Value
            self.Plot("MorningDojiStar", "morningdojistar", self.morningdojistar.Current.Value)
```

The following reference table describes the `MorningDojiStar` method:

INDICATORS

MorningDojiStar() 1/1

```
MorningDojiStar QuantConnect.Algorithm.CandlestickPatterns.MorningDojiStar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `MorningDojiStar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`MorningDojiStar` - The new `MorningDojiStar` indicator object.

Definition at line 726 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MorningDojiStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MorningDojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.morningdojistar = MorningDojiStar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.morningdojistar.Update(bar)

        if self.morningdojistar.IsReady:
            # The current value of self.morningdojistar is represented by
self.morningdojistar.Current.Value
            self.Plot("MorningDojiStar", "morningdojistar", self.morningdojistar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class MorningDojiStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.morningdojistar = MorningDojiStar()
        self.RegisterIndicator(self.symbol, self.morningdojistar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.morningdojistar.IsReady:
            # The current value of self.morningdojistar is represented by
self.morningdojistar.Current.Value
            self.Plot("MorningDojiStar", "morningdojistar", self.morningdojistar.Current.Value)
```

The following reference table describes the `MorningDojiStar` constructor:

INDICATORS

MorningDojiStar() 1/3

```
MorningDojiStar QuantConnect.Indicators.CandlestickPatterns.MorningDojiStar (
string name,
*decimal penetration
)
```

Initializes a new instance of the `MorningDojiSta` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	penetration	*(Optional) (Optional)* Percentage of penetration of a candle within another candle. Default: 0.3m.

Return

`MorningDojiStar` - The new `MorningDojiStar` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/MorningDojiStar.cs.

INDICATORS

MorningDojiStar() 2/3

```
MorningDojiStar QuantConnect.Indicators.CandlestickPatterns.MorningDojiStar (
decimal penetration
)
```

Initializes a new instance of the `MorningDojiSta` class.

Show Details

Parameters		
`decimal`	penetration	Percentage of penetration of a candle within another candle.

Return

`MorningDojiStar` - The new `MorningDojiStar` indicator object.

Definition at line 70 of file Indicators/CandlestickPatterns/MorningDojiStar.cs.

INDICATORS

MorningDojiStar() 3/3

```
MorningDojiStar QuantConnect.Indicators.CandlestickPatterns.MorningDojiStar (
)
```

Initializes a new instance of the `MorningDojiSta` class.

Show Details

This method requires no argument input.

Return

`MorningDojiStar` - The new `MorningDojiStar` indicator object.

Definition at line 78 of file Indicators/CandlestickPatterns/MorningDojiStar.cs.

Supported Indicators

Morning Star

Introduction

Create a new Morning Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MorningStar Indicator

To create an automatic indicators for `MorningStar` , call the `MorningStar` helper method from the `QCAlgorithm` class. The `MorningStar` method creates a `MorningStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MorningStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.morningstar = self.MorningStar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.morningstar.IsReady:
            # The current value of self.morningstar is represented by self.morningstar.Current.Value
            self.Plot("MorningStar", "morningstar", self.morningstar.Current.Value)
```

The following reference table describes the `MorningStar` method:

INDICATORS

MorningStar() 1/1

```
MorningStar QuantConnect.Algorithm.CandlestickPatterns.MorningStar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `MorningStar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`MorningStar` - The new `MorningStar` indicator object.

Definition at line 743 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MorningStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MorningStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.morningstar = MorningStar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.morningstar.Update(bar)

        if self.morningstar.IsReady:
            # The current value of self.morningstar is represented by self.morningstar.Current.Value
            self.Plot("MorningStar", "morningstar", self.morningstar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MorningStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.morningstar = MorningStar()
        self.RegisterIndicator(self.symbol, self.morningstar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.morningstar.IsReady:
            # The current value of self.morningstar is represented by self.morningstar.Current.Value
            self.Plot("MorningStar", "morningstar", self.morningstar.Current.Value)
```

The following reference table describes the `MorningStar` constructor:

INDICATORS

MorningStar() 1/3

```
MorningStar QuantConnect.Indicators.CandlestickPatterns.MorningStar (
string name,
*decimal penetration
)
```

Initializes a new instance of the `MorningSta` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	penetration	*(Optional) (Optional)* Percentage of penetration of a candle within another candle. Default: 0.3m.

Return

`MorningStar` - The new `MorningStar` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/MorningStar.cs.

INDICATORS

MorningStar() 2/3

```
MorningStar QuantConnect.Indicators.CandlestickPatterns.MorningStar (
decimal penetration
)
```

Initializes a new instance of the `MorningSta` class.

Show Details

Parameters		
`decimal`	penetration	Percentage of penetration of a candle within another candle.

Return

`MorningStar` - The new `MorningStar` indicator object.

Definition at line 67 of file Indicators/CandlestickPatterns/MorningStar.cs.

INDICATORS

MorningStar() 3/3

```
MorningStar QuantConnect.Indicators.CandlestickPatterns.MorningStar (
)
```

Initializes a new instance of the `MorningSta` class.

Show Details

This method requires no argument input.

Return

`MorningStar` - The new `MorningStar` indicator object.

Definition at line 75 of file Indicators/CandlestickPatterns/MorningStar.cs.

Supported Indicators

On Neck

Introduction

Create a new On-Neck candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using OnNeck Indicator

To create an automatic indicators for `OnNeck` , call the `OnNeck` helper method from the `QCAlgorithm` class. The `OnNeck` method creates a `OnNeck` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class OnNeckAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.onneck = self.OnNeck(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.onneck.IsReady:
            # The current value of self.onneck is represented by self.onneck.Current.Value
            self.Plot("OnNeck", "onneck", self.onneck.Current.Value)
```

The following reference table describes the `OnNeck` method:

INDICATORS

OnNeck() 1/1

```
OnNeck QuantConnect.Algorithm.CandlestickPatterns.OnNeck (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `OnNeck` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`OnNeck` - The new `OnNeck` indicator object.

Definition at line 759 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `OnNeck` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class OnNeckAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.onneck = OnNeck()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.onneck.Update(bar)

        if self.onneck.IsReady:
            # The current value of self.onneck is represented by self.onneck.Current.Value
            self.Plot("OnNeck", "onneck", self.onneck.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class OnNeckAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.onneck = OnNeck()
        self.RegisterIndicator(self.symbol, self.onneck, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.onneck.IsReady:
            # The current value of self.onneck is represented by self.onneck.Current.Value
            self.Plot("OnNeck", "onneck", self.onneck.Current.Value)
```

The following reference table describes the `OnNeck` constructor:

INDICATORS

OnNeck() 1/2

```
OnNeck QuantConnect.Indicators.CandlestickPatterns.OnNeck (
string name
)
```

Initializes a new instance of the `OnNec` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`OnNeck` - The new `OnNeck` indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/OnNeck.cs.

INDICATORS

OnNeck() 2/2

```
OnNeck QuantConnect.Indicators.CandlestickPatterns.OnNeck (

)
```

Initializes a new instance of the `OnNec` class.

Show Details

This method requires no argument input.

Return

`OnNeck` - The new `OnNeck` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/OnNeck.cs.

Supported Indicators

Piercing

Introduction

Create a new Piercing candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Piercing Indicator

To create an automatic indicators for `Piercing` , call the `Piercing` helper method from the `QCAlgorithm` class. The `Piercing` method creates a `Piercing` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class PiercingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.piercing = self.Piercing(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.piercing.IsReady:
            # The current value of self.piercing is represented by self.piercing.Current.Value
            self.Plot("Piercing", "piercing", self.piercing.Current.Value)
```

The following reference table describes the `Piercing` method:

INDICATORS

Piercing() 1/1

```
Piercing QuantConnect.Algorithm.CandlestickPatterns.Piercing (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Piercing` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Piercing` - The new `Piercing` indicator object.

Definition at line 775 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Piercing` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class PiercingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.piercing = Piercing()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.piercing.Update(bar)

        if self.piercing.IsReady:
            # The current value of self.piercing is represented by self.piercing.Current.Value
            self.Plot("Piercing", "piercing", self.piercing.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class PiercingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.piercing = Piercing()
        self.RegisterIndicator(self.symbol, self.piercing, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.piercing.IsReady:
            # The current value of self.piercing is represented by self.piercing.Current.Value
            self.Plot("Piercing", "piercing", self.piercing.Current.Value)
```

The following reference table describes the `Piercing` constructor:

INDICATORS

Piercing() 1/2

```
Piercing QuantConnect.Indicators.CandlestickPatterns.Piercing (
string name
)
```

Initializes a new instance of the `Piercin` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Piercing` - The new `Piercing` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/Piercing.cs.

INDICATORS

Piercing() 2/2

```
Piercing QuantConnect.Indicators.CandlestickPatterns.Piercing (

)
```

Initializes a new instance of the `Piercin` class.

Show Details

This method requires no argument input.

Return

`Piercing` - The new `Piercing` indicator object.

Definition at line 52 of file Indicators/CandlestickPatterns/Piercing.cs.

Supported Indicators

Rickshaw Man

Introduction

Create a new Rickshaw Man candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using RickshawMan Indicator

To create an automatic indicators for `RickshawMan` , call the `RickshawMan` helper method from the `QCAlgorithm` class. The `RickshawMan` method creates a `RickshawMan` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class RickshawManAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rickshawman = self.RickshawMan(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.rickshawman.IsReady:
            # The current value of self.rickshawman is represented by self.rickshawman.Current.Value
            self.Plot("RickshawMan", "rickshawman", self.rickshawman.Current.Value)
```

The following reference table describes the `RickshawMan` method:

INDICATORS

RickshawMan() 1/1

```
RickshawMan QuantConnect.Algorithm.CandlestickPatterns.RickshawMan (
 Symbol symbol,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new `RickshawMan` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`RickshawMan` - The new `RickshawMan` indicator object.

Definition at line 791 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RickshawMan` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class RickshawManAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rickshawman = RickshawMan()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rickshawman.Update(bar)

        if self.rickshawman.IsReady:
            # The current value of self.rickshawman is represented by self.rickshawman.Current.Value
            self.Plot("RickshawMan", "rickshawman", self.rickshawman.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class RickshawManAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rickshawman = RickshawMan()
        self.RegisterIndicator(self.symbol, self.rickshawman, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rickshawman.IsReady:
            # The current value of self.rickshawman is represented by self.rickshawman.Current.Value
            self.Plot("RickshawMan", "rickshawman", self.rickshawman.Current.Value)
```

The following reference table describes the `RickshawMan` constructor:

INDICATORS

RickshawMan() 1/2

```
RickshawMan QuantConnect.Indicators.CandlestickPatterns.RickshawMan (
string name
)
```

Initializes a new instance of the `RickshawMa` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`RickshawMan` - The new `RickshawMan` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/RickshawMan.cs.

INDICATORS

RickshawMan() 2/2

```
RickshawMan QuantConnect.Indicators.CandlestickPatterns.RickshawMan (

)
```

Initializes a new instance of the `RickshawMa` class.

Show Details

This method requires no argument input.

Return

`RickshawMan` - The new `RickshawMan` indicator object.

Definition at line 59 of file Indicators/CandlestickPatterns/RickshawMan.cs.

Supported Indicators

Rise Fall Three Methods

Introduction

Create a new Rising/Falling Three Methods candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using RiseFallThreeMethods Indicator

To create an automatic indicators for `RiseFallThreeMethods` , call the `RiseFallThreeMethods` helper method from the `QCAlgorithm` class. The `RiseFallThreeMethods` method creates a `RiseFallThreeMethods` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class RiseFallThreeMethodsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.risefallthreemethods = self.RiseFallThreeMethods(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.risefallthreemethods.IsReady:
            # The current value of self.risefallthreemethods is represented by
self.risefallthreemethods.Current.Value
            self.Plot("RiseFallThreeMethods", "risefallthreemethods",
self.risefallthreemethods.Current.Value)
```

The following reference table describes the `RiseFallThreeMethods` method:

INDICATORS

RiseFallThreeMethods() 1/1

```
RiseFallThreeMethods QuantConnect.Algorithm.CandlestickPatterns.RiseFallThreeMethods (
 Symbol symbol,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new `RiseFallThreeMethods` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`RiseFallThreeMethods` - The new `RiseFallThreeMethods` indicator object.

Definition at line 807 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RiseFallThreeMethods` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class RiseFallThreeMethodsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.risefallthreemethods = RiseFallThreeMethods()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.risefallthreemethods.Update(bar)

        if self.risefallthreemethods.IsReady:
            # The current value of self.risefallthreemethods is represented by
self.risefallthreemethods.Current.Value
            self.Plot("RiseFallThreeMethods", "risefallthreemethods",
self.risefallthreemethods.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class RiseFallThreeMethodsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.risefallthreemethods = RiseFallThreeMethods()
        self.RegisterIndicator(self.symbol, self.risefallthreemethods, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.risefallthreemethods.IsReady:
            # The current value of self.risefallthreemethods is represented by
self.risefallthreemethods.Current.Value
            self.Plot("RiseFallThreeMethods", "risefallthreemethods",
self.risefallthreemethods.Current.Value)
```

The following reference table describes the `RiseFallThreeMethods` constructor:

INDICATORS

RiseFallThreeMethods() 1/2

```
RiseFallThreeMethods QuantConnect.Indicators.CandlestickPatterns.RiseFallThreeMethods (
string name
)
```

Initializes a new instance of the `RiseFallThreeMethod` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`RiseFallThreeMethods` - The new `RiseFallThreeMethods` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/RiseFallThreeMethods.cs.

INDICATORS

RiseFallThreeMethods() 2/2

```
RiseFallThreeMethods QuantConnect.Indicators.CandlestickPatterns.RiseFallThreeMethods (

)
```

Initializes a new instance of the `RiseFallThreeMethod` class.

Show Details

This method requires no argument input.

Return

`RiseFallThreeMethods` - The new `RiseFallThreeMethods` indicator object.

Definition at line 57 of file Indicators/CandlestickPatterns/RiseFallThreeMethods.cs.

Supported Indicators

Separating Lines

Introduction

Create a new Separating Lines candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using SeparatingLines Indicator

To create an automatic indicators for `SeparatingLines` , call the `SeparatingLines` helper method from the `QCAlgorithm` class. The `SeparatingLines` method creates a `SeparatingLines` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class SeparatingLinesAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.separatinglines = self.SeparatingLines(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.separatinglines.IsReady:
            # The current value of self.separatinglines is represented by
self.separatinglines.Current.Value
            self.Plot("SeparatingLines", "separatinglines", self.separatinglines.Current.Value)
```

The following reference table describes the `SeparatingLines` method:

INDICATORS

SeparatingLines() 1/1

```
SeparatingLines QuantConnect.Algorithm.CandlestickPatterns.SeparatingLines (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `SeparatingLines` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`SeparatingLines` - The new `SeparatingLines` indicator object.

Definition at line 823 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `SeparatingLines` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class SeparatingLinesAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.separatinglines = SeparatingLines()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.separatinglines.Update(bar)

        if self.separatinglines.IsReady:
            # The current value of self.separatinglines is represented by
self.separatinglines.Current.Value
            self.Plot("SeparatingLines", "separatinglines", self.separatinglines.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class SeparatingLinesAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.separatinglines = SeparatingLines()
        self.RegisterIndicator(self.symbol, self.separatinglines, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.separatinglines.IsReady:
            # The current value of self.separatinglines is represented by
self.separatinglines.Current.Value
            self.Plot("SeparatingLines", "separatinglines", self.separatinglines.Current.Value)
```

The following reference table describes the `SeparatingLines` constructor:

INDICATORS

SeparatingLines() 1/2

```
SeparatingLines QuantConnect.Indicators.CandlestickPatterns.SeparatingLines (
string name
)
```

Initializes a new instance of the `SeparatingLine` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`SeparatingLines` - The new `SeparatingLines` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/SeparatingLines.cs.

INDICATORS

SeparatingLines() 2/2

```
SeparatingLines QuantConnect.Indicators.CandlestickPatterns.SeparatingLines (
)
```

Initializes a new instance of the `SeparatingLine` class.

Show Details

This method requires no argument input.

Return

`SeparatingLines` - The new `SeparatingLines` indicator object.

Definition at line 59 of file Indicators/CandlestickPatterns/SeparatingLines.cs.

Supported Indicators

Shooting Star

Introduction

Create a new Shooting Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ShootingStar Indicator

To create an automatic indicators for `ShootingStar` , call the `ShootingStar` helper method from the `QCAlgorithm` class. The `ShootingStar` method creates a `ShootingStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ShootingStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.shootingstar = self.ShootingStar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.shootingstar.IsReady:
            # The current value of self.shootingstar is represented by self.shootingstar.Current.Value
            self.Plot("ShootingStar", "shootingstar", self.shootingstar.Current.Value)
```

The following reference table describes the `ShootingStar` method:

INDICATORS

ShootingStar() 1/1

```
ShootingStar QuantConnect.Algorithm.CandlestickPatterns.ShootingStar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ShootingStar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ShootingStar` - The new `ShootingStar` indicator object.

Definition at line 839 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ShootingStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ShootingStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.shootingstar = ShootingStar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.shootingstar.Update(bar)

        if self.shootingstar.IsReady:
            # The current value of self.shootingstar is represented by self.shootingstar.Current.Value
            self.Plot("ShootingStar", "shootingstar", self.shootingstar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class ShootingStarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.shootingstar = ShootingStar()
        self.RegisterIndicator(self.symbol, self.shootingstar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.shootingstar.IsReady:
            # The current value of self.shootingstar is represented by self.shootingstar.Current.Value
            self.Plot("ShootingStar", "shootingstar", self.shootingstar.Current.Value)
```

The following reference table describes the `ShootingStar` constructor:

INDICATORS

ShootingStar() 1/2

```
ShootingStar QuantConnect.Indicators.CandlestickPatterns.ShootingStar (
string name
)
```

Initializes a new instance of the `ShootingSta` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`ShootingStar` - The new `ShootingStar` indicator object.

Definition at line 49 of file Indicators/CandlestickPatterns/ShootingStar.cs.

INDICATORS

ShootingStar() 2/2

```
ShootingStar QuantConnect.Indicators.CandlestickPatterns.ShootingStar (

)
```

Initializes a new instance of the `ShootingSta` class.

Show Details

This method requires no argument input.

Return

`ShootingStar` - The new `ShootingStar` indicator object.

Definition at line 61 of file Indicators/CandlestickPatterns/ShootingStar.cs.

Supported Indicators

Short Line Candle

Introduction

Create a new Short Line Candle candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ShortLineCandle Indicator

To create an automatic indicators for `ShortLineCandle` , call the `ShortLineCandle` helper method from the `QCAlgorithm` class. The `ShortLineCandle` method creates a `ShortLineCandle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ShortLineCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.shortlinecandle = self.ShortLineCandle(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.shortlinecandle.IsReady:
            # The current value of self.shortlinecandle is represented by
self.shortlinecandle.Current.Value
            self.Plot("ShortLineCandle", "shortlinecandle", self.shortlinecandle.Current.Value)
```

The following reference table describes the `ShortLineCandle` method:

INDICATORS

ShortLineCandle() 1/1

```
ShortLineCandle QuantConnect.Algorithm.CandlestickPatterns.ShortLineCandle (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ShortLineCandle` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ShortLineCandle` - The new `ShortLineCandle` indicator object.

Definition at line 855 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ShortLineCandle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ShortLineCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.shortlinecandle = ShortLineCandle()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.shortlinecandle.Update(bar)

        if self.shortlinecandle.IsReady:
            # The current value of self.shortlinecandle is represented by
self.shortlinecandle.Current.Value
            self.Plot("ShortLineCandle", "shortlinecandle", self.shortlinecandle.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ShortLineCandleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.shortlinecandle = ShortLineCandle()
        self.RegisterIndicator(self.symbol, self.shortlinecandle, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.shortlinecandle.IsReady:
            # The current value of self.shortlinecandle is represented by
self.shortlinecandle.Current.Value
            self.Plot("ShortLineCandle", "shortlinecandle", self.shortlinecandle.Current.Value)
```

The following reference table describes the `ShortLineCandle` constructor:

INDICATORS

ShortLineCandle() 1/2

```
ShortLineCandle QuantConnect.Indicators.CandlestickPatterns.ShortLineCandle (
string name
)
```

Initializes a new instance of the `ShortLineCandl` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`ShortLineCandle` - The new `ShortLineCandle` indicator object.

Definition at line 44 of file Indicators/CandlestickPatterns/ShortLineCandle.cs.

INDICATORS

ShortLineCandle() 2/2

```
ShortLineCandle QuantConnect.Indicators.CandlestickPatterns.ShortLineCandle (
)
```

Initializes a new instance of the `ShortLineCandl` class.

Show Details

This method requires no argument input.

Return

`ShortLineCandle` - The new `ShortLineCandle` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/ShortLineCandle.cs.

Supported Indicators

Spinning Top

Introduction

Create a new Spinning Top candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using SpinningTop Indicator

To create an automatic indicators for `SpinningTop` , call the `SpinningTop` helper method from the `QCAlgorithm` class. The `SpinningTop` method creates a `SpinningTop` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class SpinningTopAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.spinningtop = self.SpinningTop(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.spinningtop.IsReady:
            # The current value of self.spinningtop is represented by self.spinningtop.Current.Value
            self.Plot("SpinningTop", "spinningtop", self.spinningtop.Current.Value)
```

The following reference table describes the `SpinningTop` method:

INDICATORS

SpinningTop() 1/1

```
SpinningTop QuantConnect.Algorithm.CandlestickPatterns.SpinningTop (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `SpinningTop` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`SpinningTop` - The new `SpinningTop` indicator object.

Definition at line 871 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `SpinningTop` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class SpinningTopAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.spinningtop = SpinningTop()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.spinningtop.Update(bar)

        if self.spinningtop.IsReady:
            # The current value of self.spinningtop is represented by self.spinningtop.Current.Value
            self.Plot("SpinningTop", "spinningtop", self.spinningtop.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SpinningTopAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.spinningtop = SpinningTop()
        self.RegisterIndicator(self.symbol, self.spinningtop, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.spinningtop.IsReady:
            # The current value of self.spinningtop is represented by self.spinningtop.Current.Value
            self.Plot("SpinningTop", "spinningtop", self.spinningtop.Current.Value)
```

The following reference table describes the SpinningTop constructor:

INDICATORS

SpinningTop() 1/2

```
SpinningTop QuantConnect.Indicators.CandlestickPatterns.SpinningTop (
string name
)
```

Initializes a new instance of the SpinningTo class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

SpinningTop - The new SpinningTop indicator object.

Definition at line 41 of file Indicators/CandlestickPatterns/SpinningTop.cs.

INDICATORS

SpinningTop() 2/2

```
SpinningTop QuantConnect.Indicators.CandlestickPatterns.SpinningTop (
)
```

Initializes a new instance of the `SpinningTo` class.

Show Details

This method requires no argument input.

Return

`SpinningTop` - The new `SpinningTop` indicator object.

Definition at line 50 of file Indicators/CandlestickPatterns/SpinningTop.cs.

Supported Indicators

Stalled Pattern

Introduction

Create a new Stalled Pattern candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using StalledPattern Indicator

To create an automatic indicators for `StalledPattern` , call the `StalledPattern` helper method from the `QCAlgorithm` class. The `StalledPattern` method creates a `StalledPattern` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class StalledPatternAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.stalledpattern = self.StalledPattern(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.stalledpattern.IsReady:
            # The current value of self.stalledpattern is represented by self.stalledpattern.Current.Value
            self.Plot("StalledPattern", "stalledpattern", self.stalledpattern.Current.Value)
```

The following reference table describes the `StalledPattern` method:

INDICATORS

StalledPattern() 1/1

```
StalledPattern QuantConnect.Algorithm.CandlestickPatterns.StalledPattern (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `StalledPattern` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`StalledPattern` - The new `StalledPattern` indicator object.

Definition at line 887 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `StalledPattern` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```
class StalledPatternAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.stalledpattern = StalledPattern()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.stalledpattern.Update(bar)

        if self.stalledpattern.IsReady:
            # The current value of self.stalledpattern is represented by self.stalledpattern.Current.Value
            self.Plot("StalledPattern", "stalledpattern", self.stalledpattern.Current.Value)
```

PY

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class StalledPatternAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.stalledpattern = StalledPattern()
        self.RegisterIndicator(self.symbol, self.stalledpattern, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.stalledpattern.IsReady:
            # The current value of self.stalledpattern is represented by self.stalledpattern.Current.Value
            self.Plot("StalledPattern", "stalledpattern", self.stalledpattern.Current.Value)
```

The following reference table describes the `StalledPattern` constructor:

INDICATORS

StalledPattern() 1/2

```
StalledPattern QuantConnect.Indicators.CandlestickPatterns.StalledPattern (
string name
)
```

Initializes a new instance of the `StalledPatter` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`StalledPattern` - The new `StalledPattern` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/StalledPattern.cs.

INDICATORS

StalledPattern() 2/2

```
StalledPattern QuantConnect.Indicators.CandlestickPatterns.StalledPattern (
)
```

Initializes a new instance of the `StalledPatter` class.

Show Details

This method requires no argument input.

Return

`StalledPattern` - The new `StalledPattern` indicator object.

Definition at line 67 of file Indicators/CandlestickPatterns/StalledPattern.cs.

Supported Indicators

Stick Sandwich

Introduction

Create a new Stick Sandwich candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using StickSandwich Indicator

To create an automatic indicators for `StickSandwich` , call the `StickSandwich` helper method from the `QCAlgorithm` class. The `StickSandwich` method creates a `StickSandwich` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class StickSandwichAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sticksandwich = self.StickSandwich(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.sticksandwich.IsReady:
            # The current value of self.sticksandwich is represented by self.sticksandwich.Current.Value
            self.Plot("StickSandwich", "sticksandwich", self.sticksandwich.Current.Value)
```

The following reference table describes the `StickSandwich` method:

INDICATORS

StickSandwich() 1/1

```
StickSandwich QuantConnect.Algorithm.CandlestickPatterns.StickSandwich (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `StickSandwich` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`StickSandwich` - The new `StickSandwich` indicator object.

Definition at line 903 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `StickSandwich` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class StickSandwichAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sticksandwich = StickSandwich()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.sticksandwich.Update(bar)

        if self.sticksandwich.IsReady:
            # The current value of self.sticksandwich is represented by self.sticksandwich.Current.Value
            self.Plot("StickSandwich", "sticksandwich", self.sticksandwich.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class StickSandwichAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sticksandwich = StickSandwich()
        self.RegisterIndicator(self.symbol, self.sticksandwich, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.sticksandwich.IsReady:
            # The current value of self.sticksandwich is represented by self.sticksandwich.Current.Value
            self.Plot("StickSandwich", "sticksandwich", self.sticksandwich.Current.Value)
```

The following reference table describes the `StickSandwich` constructor:

INDICATORS

StickSandwich() 1/2

```
StickSandwich QuantConnect.Indicators.CandlestickPatterns.StickSandwich (
string name
)
```

Initializes a new instance of the `StickSandwic` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`StickSandwich` - The new `StickSandwich` indicator object.

Definition at line 43 of file Indicators/CandlestickPatterns/StickSandwich.cs.

INDICATORS

StickSandwich() 2/2

```
StickSandwich QuantConnect.Indicators.CandlestickPatterns.StickSandwich (

)
```

Initializes a new instance of the `StickSandwic` class.

Show Details

This method requires no argument input.

Return

`StickSandwich` - The new `StickSandwich` indicator object.

Definition at line 52 of file Indicators/CandlestickPatterns/StickSandwich.cs.

Supported Indicators

Takuri

Introduction

Create a new Takuri (Dragonfly Doji with very long lower shadow) candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Takuri Indicator

To create an automatic indicators for `Takuri` , call the `Takuri` helper method from the `QCAlgorithm` class. The `Takuri` method creates a `Takuri` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TakuriAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.takuri = self.Takuri(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.takuri.IsReady:
            # The current value of self.takuri is represented by self.takuri.Current.Value
            self.Plot("Takuri", "takuri", self.takuri.Current.Value)
```

The following reference table describes the `Takuri` method:

INDICATORS

Takuri() 1/1

```
Takuri QuantConnect.Algorithm.CandlestickPatterns.Takuri (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Takuri` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Takuri` - The new `Takuri` indicator object.

Definition at line 919 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Takuri` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class TakuriAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.takuri = Takuri()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.takuri.Update(bar)

        if self.takuri.IsReady:
            # The current value of self.takuri is represented by self.takuri.Current.Value
            self.Plot("Takuri", "takuri", self.takuri.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class TakuriAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.takuri = Takuri()
        self.RegisterIndicator(self.symbol, self.takuri, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.takuri.IsReady:
            # The current value of self.takuri is represented by self.takuri.Current.Value
            self.Plot("Takuri", "takuri", self.takuri.Current.Value)
```

The following reference table describes the `Takuri` constructor:

INDICATORS

Takuri() 1/2

```
Takuri QuantConnect.Indicators.CandlestickPatterns.Takuri (
string name
)
```

Initializes a new instance of the `Takur` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Takuri` - The new `Takuri` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/Takuri.cs.

INDICATORS

Takuri() 2/2

```
Takuri QuantConnect.Indicators.CandlestickPatterns.Takuri (

)
```

Initializes a new instance of the `Takur` class.

Show Details

This method requires no argument input.

Return

`Takuri` - The new `Takuri` indicator object.

Definition at line 59 of file Indicators/CandlestickPatterns/Takuri.cs.

Supported Indicators

Tasuki Gap

Introduction

Create a new Tasuki Gap candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using TasukiGap Indicator

To create an automatic indicators for `TasukiGap` , call the `TasukiGap` helper method from the `QCAlgorithm` class. The `TasukiGap` method creates a `TasukiGap` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TasukiGapAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tasukigap = self.TasukiGap(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.tasukigap.IsReady:
            # The current value of self.tasukigap is represented by self.tasukigap.Current.Value
            self.Plot("TasukiGap", "tasukigap", self.tasukigap.Current.Value)
```

The following reference table describes the `TasukiGap` method:

INDICATORS

TasukiGap() 1/1

```
TasukiGap QuantConnect.Algorithm.CandlestickPatterns.TasukiGap (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `TasukiGap` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`TasukiGap` - The new `TasukiGap` indicator object.

Definition at line 935 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TasukiGap` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class TasukiGapAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tasukigap = TasukiGap()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tasukigap.Update(bar)

        if self.tasukigap.IsReady:
            # The current value of self.tasukigap is represented by self.tasukigap.Current.Value
            self.Plot("TasukiGap", "tasukigap", self.tasukigap.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class TasukiGapAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tasukigap = TasukiGap()
        self.RegisterIndicator(self.symbol, self.tasukigap, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tasukigap.IsReady:
            # The current value of self.tasukigap is represented by self.tasukigap.Current.Value
            self.Plot("TasukiGap", "tasukigap", self.tasukigap.Current.Value)
```

The following reference table describes the TasukiGap constructor:

INDICATORS

TasukiGap() 1/2

```
TasukiGap QuantConnect.Indicators.CandlestickPatterns.TasukiGap (
string name
)
```

Initializes a new instance of the TasukiGa class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

TasukiGap - The new TasukiGap indicator object.

Definition at line 46 of file Indicators/CandlestickPatterns/TasukiGap.cs.

INDICATORS

TasukiGap() 2/2

```
TasukiGap QuantConnect.Indicators.CandlestickPatterns.TasukiGap (

)
```

Initializes a new instance of the `TasukiGa` class.

Show Details

This method requires no argument input.

Return

`TasukiGap` - The new `TasukiGap` indicator object.

Definition at line 55 of file Indicators/CandlestickPatterns/TasukiGap.cs.

Supported Indicators

Three Black Crows

Introduction

Create a new Three Black Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeBlackCrows Indicator

To create an automatic indicators for `ThreeBlackCrows` , call the `ThreeBlackCrows` helper method from the `QCAlgorithm` class. The `ThreeBlackCrows` method creates a `ThreeBlackCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ThreeBlackCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeblackcrows = self.ThreeBlackCrows(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.threeblackcrows.IsReady:
            # The current value of self.threeblackcrows is represented by
self.threeblackcrows.Current.Value
            self.Plot("ThreeBlackCrows", "threeblackcrows", self.threeblackcrows.Current.Value)
```

The following reference table describes the `ThreeBlackCrows` method:

INDICATORS

ThreeBlackCrows() 1/1

```
ThreeBlackCrows QuantConnect.Algorithm.CandlestickPatterns.ThreeBlackCrows (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ThreeBlackCrows` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ThreeBlackCrows` - The new `ThreeBlackCrows` indicator object.

Definition at line 64 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ThreeBlackCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThreeBlackCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeblackcrows = ThreeBlackCrows()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.threeblackcrows.Update(bar)

        if self.threeblackcrows.IsReady:
            # The current value of self.threeblackcrows is represented by
self.threeblackcrows.Current.Value
            self.Plot("ThreeBlackCrows", "threeblackcrows", self.threeblackcrows.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ThreeBlackCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeblackcrows = ThreeBlackCrows()
        self.RegisterIndicator(self.symbol, self.threeblackcrows, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.threeblackcrows.IsReady:
            # The current value of self.threeblackcrows is represented by
self.threeblackcrows.Current.Value
            self.Plot("ThreeBlackCrows", "threeblackcrows", self.threeblackcrows.Current.Value)
```

The following reference table describes the `ThreeBlackCrows` constructor:

INDICATORS

ThreeBlackCrows() 1/2

```
ThreeBlackCrows QuantConnect.Indicators.CandlestickPatterns.ThreeBlackCrows (
string name
)
```

Initializes a new instance of the `ThreeBlackCrow` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`ThreeBlackCrows` - The new `ThreeBlackCrows` indicator object.

Definition at line 44 of file Indicators/CandlestickPatterns/ThreeBlackCrows.cs.

INDICATORS

ThreeBlackCrows() 2/2

```
ThreeBlackCrows QuantConnect.Indicators.CandlestickPatterns.ThreeBlackCrows (
)
```

Initializes a new instance of the `ThreeBlackCrow` class.

Show Details

This method requires no argument input.

Return

`ThreeBlackCrows` - The new `ThreeBlackCrows` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/ThreeBlackCrows.cs.

Supported Indicators

Three Inside

Introduction

Create a new Three Inside Up/Down candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeInside Indicator

To create an automatic indicators for `ThreeInside` , call the `ThreeInside` helper method from the `QCAlgorithm` class. The `ThreeInside` method creates a `ThreeInside` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ThreeInsideAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeinside = self.ThreeInside(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.threeinside.IsReady:
            # The current value of self.threeinside is represented by self.threeinside.Current.Value
            self.Plot("ThreeInside", "threeinside", self.threeinside.Current.Value)
```

The following reference table describes the `ThreeInside` method:

INDICATORS

ThreeInside() 1/1

```
ThreeInside QuantConnect.Algorithm.CandlestickPatterns.ThreeInside (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ThreeInside` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ThreeInside` - The new `ThreeInside` indicator object.

Definition at line 80 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ThreeInside` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThreeInsideAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeinside = ThreeInside()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.threeinside.Update(bar)

        if self.threeinside.IsReady:
            # The current value of self.threeinside is represented by self.threeinside.Current.Value
            self.Plot("ThreeInside", "threeinside", self.threeinside.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ThreeInsideAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeinside = ThreeInside()
        self.RegisterIndicator(self.symbol, self.threeinside, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.threeinside.IsReady:
            # The current value of self.threeinside is represented by self.threeinside.Current.Value
            self.Plot("ThreeInside", "threeinside", self.threeinside.Current.Value)
```

The following reference table describes the `ThreeInside` constructor:

INDICATORS

ThreeInside() 1/2

```
ThreeInside QuantConnect.Indicators.CandlestickPatterns.ThreeInside (
string name
)
```

Initializes a new instance of the `ThreeInsid` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`ThreeInside` - The new `ThreeInside` indicator object.

Definition at line 46 of file Indicators/CandlestickPatterns/ThreeInside.cs.

INDICATORS

ThreeInside() 2/2

```
ThreeInside QuantConnect.Indicators.CandlestickPatterns.ThreeInside (
)
```

Initializes a new instance of the `ThreeInsid` class.

Show Details

This method requires no argument input.

Return

`ThreeInside` - The new `ThreeInside` indicator object.

Definition at line 56 of file Indicators/CandlestickPatterns/ThreeInside.cs.

Supported Indicators

Three Line Strike

Introduction

Create a new Three Line Strike candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeLineStrike Indicator

To create an automatic indicators for `ThreeLineStrike` , call the `ThreeLineStrike` helper method from the `QCAlgorithm` class. The `ThreeLineStrike` method creates a `ThreeLineStrike` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class ThreeLineStrikeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threelinestrike = self.ThreeLineStrike(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.threelinestrike.IsReady:
            # The current value of self.threelinestrike is represented by
self.threelinestrike.Current.Value
            self.Plot("ThreeLineStrike", "threelinestrike", self.threelinestrike.Current.Value)
```

The following reference table describes the `ThreeLineStrike` method:

INDICATORS

ThreeLineStrike() 1/1

```
ThreeLineStrike QuantConnect.Algorithm.CandlestickPatterns.ThreeLineStrike (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ThreeLineStrike` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ThreeLineStrike` - The new `ThreeLineStrike` indicator object.

Definition at line 96 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ThreeLineStrike` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThreeLineStrikeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threelinestrike = ThreeLineStrike()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.threelinestrike.Update(bar)

        if self.threelinestrike.IsReady:
            # The current value of self.threelinestrike is represented by
self.threelinestrike.Current.Value
            self.Plot("ThreeLineStrike", "threelinestrike", self.threelinestrike.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ThreeLineStrikeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threelinestrike = ThreeLineStrike()
        self.RegisterIndicator(self.symbol, self.threelinestrike, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.threelinestrike.IsReady:
            # The current value of self.threelinestrike is represented by
self.threelinestrike.Current.Value
            self.Plot("ThreeLineStrike", "threelinestrike", self.threelinestrike.Current.Value)
```

The following reference table describes the ThreeLineStrike constructor:

INDICATORS

ThreeLineStrike() 1/2

```
ThreeLineStrike QuantConnect.Indicators.CandlestickPatterns.ThreeLineStrike (
string name
)
```

Initializes a new instance of the ThreeLineStrik class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

ThreeLineStrike - The new ThreeLineStrike indicator object.

Definition at line 45 of file Indicators/CandlestickPatterns/ThreeLineStrike.cs.

INDICATORS

ThreeLineStrike() 2/2

```
ThreeLineStrike QuantConnect.Indicators.CandlestickPatterns.ThreeLineStrike (
)
```

Initializes a new instance of the `ThreeLineStrik` class.

Show Details

This method requires no argument input.

Return

`ThreeLineStrike` - The new `ThreeLineStrike` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/ThreeLineStrike.cs.

Supported Indicators

Three Outside

Introduction

Create a new Three Outside Up/Down candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeOutside Indicator

To create an automatic indicators for `ThreeOutside` , call the `ThreeOutside` helper method from the `QCAlgorithm` class. The `ThreeOutside` method creates a `ThreeOutside` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ThreeOutsideAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeoutside = self.ThreeOutside(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.threeoutside.IsReady:
            # The current value of self.threeoutside is represented by self.threeoutside.Current.Value
            self.Plot("ThreeOutside", "threeoutside", self.threeoutside.Current.Value)
```

The following reference table describes the `ThreeOutside` method:

INDICATORS

ThreeOutside() 1/1

```
ThreeOutside QuantConnect.Algorithm.CandlestickPatterns.ThreeOutside (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ThreeOutside` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ThreeOutside` - The new `ThreeOutside` indicator object.

Definition at line 112 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ThreeOutside` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThreeOutsideAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeoutside = ThreeOutside()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.threeoutside.Update(bar)

        if self.threeoutside.IsReady:
            # The current value of self.threeoutside is represented by self.threeoutside.Current.Value
            self.Plot("ThreeOutside", "threeoutside", self.threeoutside.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class ThreeOutsideAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threeoutside = ThreeOutside()
        self.RegisterIndicator(self.symbol, self.threeoutside, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.threeoutside.IsReady:
            # The current value of self.threeoutside is represented by self.threeoutside.Current.Value
            self.Plot("ThreeOutside", "threeoutside", self.threeoutside.Current.Value)
```

The following reference table describes the `ThreeOutside` constructor:

INDICATORS

ThreeOutside() 1/2

```
ThreeOutside QuantConnect.Indicators.CandlestickPatterns.ThreeOutside (
string name
)
```

Initializes a new instance of the `ThreeOutsid` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`ThreeOutside` - The new `ThreeOutside` indicator object.

Definition at line 39 of file Indicators/CandlestickPatterns/ThreeOutside.cs.

INDICATORS

ThreeOutside() 2/2

```
ThreeOutside QuantConnect.Indicators.CandlestickPatterns.ThreeOutside (
)
```

Initializes a new instance of the `ThreeOutsid` class.

Show Details

This method requires no argument input.

Return

`ThreeOutside` - The new `ThreeOutside` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/ThreeOutside.cs.

Supported Indicators

Three Stars In South

Introduction

Create a new Three Stars In The South candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeStarsInSouth Indicator

To create an automatic indicators for `ThreeStarsInSouth` , call the `ThreeStarsInSouth` helper method from the `QCAlgorithm` class. The `ThreeStarsInSouth` method creates a `ThreeStarsInSouth` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ThreeStarsInSouthAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threestarsinsouth = self.ThreeStarsInSouth(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.threestarsinsouth.IsReady:
            # The current value of self.threestarsinsouth is represented by
self.threestarsinsouth.Current.Value
            self.Plot("ThreeStarsInSouth", "threestarsinsouth", self.threestarsinsouth.Current.Value)
```

The following reference table describes the `ThreeStarsInSouth` method:

INDICATORS

ThreeStarsInSouth() 1/1

```
ThreeStarsInSouth QuantConnect.Algorithm.CandlestickPatterns.ThreeStarsInSouth (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ThreeStarsInSouth` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ThreeStarsInSouth` - The new `ThreeStarsInSouth` indicator object.

Definition at line 128 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ThreeStarsInSouth` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThreeStarsInSouthAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threestarsinsouth = ThreeStarsInSouth()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.threestarsinsouth.Update(bar)

        if self.threestarsinsouth.IsReady:
            # The current value of self.threestarsinsouth is represented by
self.threestarsinsouth.Current.Value
            self.Plot("ThreeStarsInSouth", "threestarsinsouth", self.threestarsinsouth.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ThreeStarsInSouthAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threestarsinsouth = ThreeStarsInSouth()
        self.RegisterIndicator(self.symbol, self.threestarsinsouth, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.threestarsinsouth.IsReady:
            # The current value of self.threestarsinsouth is represented by
self.threestarsinsouth.Current.Value
            self.Plot("ThreeStarsInSouth", "threestarsinsouth", self.threestarsinsouth.Current.Value)
```

The following reference table describes the ThreeStarsInSouth constructor:

INDICATORS

ThreeStarsInSouth() 1/2

```
ThreeStarsInSouth QuantConnect.Indicators.CandlestickPatterns.ThreeStarsInSouth (
string name
)
```

Initializes a new instance of the ThreeStarsInSout class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

ThreeStarsInSouth - The new ThreeStarsInSouth indicator object.

Definition at line 52 of file Indicators/CandlestickPatterns/ThreeStarsInSouth.cs.

INDICATORS

ThreeStarsInSouth() 2/2

```
ThreeStarsInSouth QuantConnect.Indicators.CandlestickPatterns.ThreeStarsInSouth (
)
```

Initializes a new instance of the `ThreeStarsInSout` class.

Show Details

This method requires no argument input.

Return

`ThreeStarsInSouth` - The new `ThreeStarsInSouth` indicator object.

Definition at line 65 of file Indicators/CandlestickPatterns/ThreeStarsInSouth.cs.

Supported Indicators

Three White Soldiers

Introduction

Create a new Three Advancing White Soldiers candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeWhiteSoldiers Indicator

To create an automatic indicators for `ThreeWhiteSoldiers` , call the `ThreeWhiteSoldiers` helper method from the `QCAlgorithm` class. The `ThreeWhiteSoldiers` method creates a `ThreeWhiteSoldiers` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class ThreeWhiteSoldiersAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threewhitesoldiers = self.ThreeWhiteSoldiers(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.threewhitesoldiers.IsReady:
            # The current value of self.threewhitesoldiers is represented by
self.threewhitesoldiers.Current.Value
            self.Plot("ThreeWhiteSoldiers", "threewhitesoldiers", self.threewhitesoldiers.Current.Value)
```

The following reference table describes the `ThreeWhiteSoldiers` method:

INDICATORS

ThreeWhiteSoldiers() 1/1

```
ThreeWhiteSoldiers QuantConnect.Algorithm.CandlestickPatterns.ThreeWhiteSoldiers (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `ThreeWhiteSoldiers` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`ThreeWhiteSoldiers` - The new `ThreeWhiteSoldiers` indicator object.

Definition at line 144 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ThreeWhiteSoldiers` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThreeWhiteSoldiersAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threewhitesoldiers = ThreeWhiteSoldiers()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.threewhitesoldiers.Update(bar)

        if self.threewhitesoldiers.IsReady:
            # The current value of self.threewhitesoldiers is represented by
self.threewhitesoldiers.Current.Value
            self.Plot("ThreeWhiteSoldiers", "threewhitesoldiers", self.threewhitesoldiers.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class ThreeWhiteSoldiersAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.threewhitesoldiers = ThreeWhiteSoldiers()
        self.RegisterIndicator(self.symbol, self.threewhitesoldiers, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.threewhitesoldiers.IsReady:
            # The current value of self.threewhitesoldiers is represented by
self.threewhitesoldiers.Current.Value
            self.Plot("ThreeWhiteSoldiers", "threewhitesoldiers", self.threewhitesoldiers.Current.Value)
```

The following reference table describes the `ThreeWhiteSoldiers` constructor:

INDICATORS

ThreeWhiteSoldiers() 1/2

```
ThreeWhiteSoldiers QuantConnect.Indicators.CandlestickPatterns.ThreeWhiteSoldiers (
string name
)
```

Initializes a new instance of the `ThreeWhiteSoldier` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`ThreeWhiteSoldiers` - The new `ThreeWhiteSoldiers` indicator object.

Definition at line 54 of file Indicators/CandlestickPatterns/ThreeWhiteSoldiers.cs.

INDICATORS

ThreeWhiteSoldiers() 2/2

```
ThreeWhiteSoldiers QuantConnect.Indicators.CandlestickPatterns.ThreeWhiteSoldiers (
)
```

Initializes a new instance of the `ThreeWhiteSoldier` class.

Show Details

This method requires no argument input.

Return

`ThreeWhiteSoldiers` - The new `ThreeWhiteSoldiers` indicator object.

Definition at line 67 of file Indicators/CandlestickPatterns/ThreeWhiteSoldiers.cs.

Supported Indicators

Thrusting

Introduction

Create a new Thrusting candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Thrusting Indicator

To create an automatic indicators for `Thrusting` , call the `Thrusting` helper method from the `QCAlgorithm` class. The `Thrusting` method creates a `Thrusting` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ThrustingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.thrusting = self.Thrusting(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.thrusting.IsReady:
            # The current value of self.thrusting is represented by self.thrusting.Current.Value
            self.Plot("Thrusting", "thrusting", self.thrusting.Current.Value)
```

The following reference table describes the `Thrusting` method:

INDICATORS

Thrusting() 1/1

```
Thrusting QuantConnect.Algorithm.CandlestickPatterns.Thrusting (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Thrusting` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Thrusting` - The new `Thrusting` indicator object.

Definition at line 951 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Thrusting` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ThrustingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.thrusting = Thrusting()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.thrusting.Update(bar)

        if self.thrusting.IsReady:
            # The current value of self.thrusting is represented by self.thrusting.Current.Value
            self.Plot("Thrusting", "thrusting", self.thrusting.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class ThrustingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.thrusting = Thrusting()
        self.RegisterIndicator(self.symbol, self.thrusting, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.thrusting.IsReady:
            # The current value of self.thrusting is represented by self.thrusting.Current.Value
            self.Plot("Thrusting", "thrusting", self.thrusting.Current.Value)
```

The following reference table describes the `Thrusting` constructor:

INDICATORS

Thrusting() 1/2

```
Thrusting QuantConnect.Indicators.CandlestickPatterns.Thrusting (
string name
)
```

Initializes a new instance of the `Thrustin` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`Thrusting` - The new `Thrusting` indicator object.

Definition at line 47 of file Indicators/CandlestickPatterns/Thrusting.cs.

INDICATORS

Thrusting() 2/2

```
Thrusting QuantConnect.Indicators.CandlestickPatterns.Thrusting (

)
```

Initializes a new instance of the `Thrustin` class.

Show Details

This method requires no argument input.

Return

`Thrusting` - The new `Thrusting` indicator object.

Definition at line 57 of file Indicators/CandlestickPatterns/Thrusting.cs.

Supported Indicators

Tristar

Introduction

Create a new Tristar candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Tristar Indicator

To create an automatic indicators for `Tristar` , call the `Tristar` helper method from the `QCAlgorithm` class. The `Tristar` method creates a `Tristar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TristarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tristar = self.Tristar(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.tristar.IsReady:
            # The current value of self.tristar is represented by self.tristar.Current.Value
            self.Plot("Tristar", "tristar", self.tristar.Current.Value)
```

The following reference table describes the `Tristar` method:

INDICATORS

Tristar() 1/1

```
Tristar QuantConnect.Algorithm.CandlestickPatterns.Tristar (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `Tristar` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`Tristar` - The new `Tristar` indicator object.

Definition at line 967 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Tristar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class TristarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tristar = Tristar()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tristar.Update(bar)

        if self.tristar.IsReady:
            # The current value of self.tristar is represented by self.tristar.Current.Value
            self.Plot("Tristar", "tristar", self.tristar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class TristarAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tristar = Tristar()
        self.RegisterIndicator(self.symbol, self.tristar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tristar.IsReady:
            # The current value of self.tristar is represented by self.tristar.Current.Value
            self.Plot("Tristar", "tristar", self.tristar.Current.Value)
```

The following reference table describes the `Tristar` constructor:

INDICATORS

Tristar() 1/2

```
Tristar QuantConnect.Indicators.CandlestickPatterns.Tristar (
string name
)
```

Initializes a new instance of the `Trista` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`Tristar` - The new `Tristar` indicator object.

Definition at line 41 of file Indicators/CandlestickPatterns/Tristar.cs.

INDICATORS

Tristar() 2/2

```
Tristar QuantConnect.Indicators.CandlestickPatterns.Tristar (

)
```

Initializes a new instance of the `Trista` class.

Show Details

This method requires no argument input.

Return

`Tristar` - The new `Tristar` indicator object.

Definition at line 50 of file Indicators/CandlestickPatterns/Tristar.cs.

Supported Indicators

Two Crows

Introduction

Create a new Two Crows candlestick pattern indicator to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using TwoCrows Indicator

To create an automatic indicators for `TwoCrows` , call the `TwoCrows` helper method from the `QCAlgorithm` class. The `TwoCrows` method creates a `TwoCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class TwoCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.twocrows = self.TwoCrows(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.twocrows.IsReady:
            # The current value of self.twocrows is represented by self.twocrows.Current.Value
            self.Plot("TwoCrows", "twocrows", self.twocrows.Current.Value)
```

The following reference table describes the `TwoCrows` method:

INDICATORS

TwoCrows() 1/1

```
TwoCrows QuantConnect.Algorithm.CandlestickPatterns.TwoCrows (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `TwoCrows` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`TwoCrows` - The new `TwoCrows` indicator object.

Definition at line 48 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TwoCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class TwoCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.twocrows = TwoCrows()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.twocrows.Update(bar)

        if self.twocrows.IsReady:
            # The current value of self.twocrows is represented by self.twocrows.Current.Value
            self.Plot("TwoCrows", "twocrows", self.twocrows.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class TwoCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.twocrows = TwoCrows()
        self.RegisterIndicator(self.symbol, self.twocrows, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.twocrows.IsReady:
            # The current value of self.twocrows is represented by self.twocrows.Current.Value
            self.Plot("TwoCrows", "twocrows", self.twocrows.Current.Value)
```

The following reference table describes the `TwoCrows` constructor:

INDICATORS

TwoCrows() 1/2

```
TwoCrows QuantConnect.Indicators.CandlestickPatterns.TwoCrows (
string name
)
```

Initializes a new instance of the `TwoCrow` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`TwoCrows` - The new `TwoCrows` indicator object.

Definition at line 44 of file Indicators/CandlestickPatterns/TwoCrows.cs.

INDICATORS

TwoCrows() 2/2

```
TwoCrows QuantConnect.Indicators.CandlestickPatterns.TwoCrows (

)
```

Initializes a new instance of the `TwoCrow` class.

Show Details

This method requires no argument input.

Return

`TwoCrows` - The new `TwoCrows` indicator object.

Definition at line 53 of file Indicators/CandlestickPatterns/TwoCrows.cs.

Supported Indicators

Unique Three River

Introduction

Create a new Unique Three River candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using UniqueThreeRiver Indicator

To create an automatic indicators for `UniqueThreeRiver` , call the `UniqueThreeRiver` helper method from the `QCAlgorithm` class. The `UniqueThreeRiver` method creates a `UniqueThreeRiver` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class UniqueThreeRiverAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.uniquethreeriver = self.UniqueThreeRiver(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.uniquethreeriver.IsReady:
            # The current value of self.uniquethreeriver is represented by
self.uniquethreeriver.Current.Value
            self.Plot("UniqueThreeRiver", "uniquethreeriver", self.uniquethreeriver.Current.Value)
```

PY

The following reference table describes the `UniqueThreeRiver` method:

INDICATORS

UniqueThreeRiver() 1/1

```
UniqueThreeRiver QuantConnect.Algorithm.CandlestickPatterns.UniqueThreeRiver (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `UniqueThreeRiver` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`UniqueThreeRiver` - The new `UniqueThreeRiver` indicator object.

Definition at line 983 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `UniqueThreeRiver` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class UniqueThreeRiverAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.uniquethreeriver = UniqueThreeRiver()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.uniquethreeriver.Update(bar)

        if self.uniquethreeriver.IsReady:
            # The current value of self.uniquethreeriver is represented by
self.uniquethreeriver.Current.Value
            self.Plot("UniqueThreeRiver", "uniquethreeriver", self.uniquethreeriver.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class UniqueThreeRiverAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.uniquethreeriver = UniqueThreeRiver()
        self.RegisterIndicator(self.symbol, self.uniquethreeriver, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.uniquethreeriver.IsReady:
            # The current value of self.uniquethreeriver is represented by
self.uniquethreeriver.Current.Value
            self.Plot("UniqueThreeRiver", "uniquethreeriver", self.uniquethreeriver.Current.Value)
```

The following reference table describes the `UniqueThreeRiver` constructor:

INDICATORS

UniqueThreeRiver() 1/2

```
UniqueThreeRiver QuantConnect.Indicators.CandlestickPatterns.UniqueThreeRiver (
string name
)
```

Initializes a new instance of the `UniqueThreeRive` class using the specified name.

Show Details

Parameters		
string	name	The name of this indicator.

Return

`UniqueThreeRiver` - The new `UniqueThreeRiver` indicator object.

Definition at line 46 of file Indicators/CandlestickPatterns/UniqueThreeRiver.cs.

INDICATORS

UniqueThreeRiver() 2/2

```
UniqueThreeRiver QuantConnect.Indicators.CandlestickPatterns.UniqueThreeRiver (
)
```

Initializes a new instance of the `UniqueThreeRive` class.

Show Details

This method requires no argument input.

Return

`UniqueThreeRiver` - The new `UniqueThreeRiver` indicator object.

Definition at line 56 of file Indicators/CandlestickPatterns/UniqueThreeRiver.cs.

Supported Indicators

Up Down Gap Three Methods

Introduction

Create a new Up/Down Gap Three Methods candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using UpDownGapThreeMethods Indicator

To create an automatic indicators for `UpDownGapThreeMethods` , call the `UpDownGapThreeMethods` helper method from the `QCAlgorithm` class. The `UpDownGapThreeMethods` method creates a `UpDownGapThreeMethods` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class UpDownGapThreeMethodsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.updowngapthreemethods = self.UpDownGapThreeMethods(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.updowngapthreemethods.IsReady:
            # The current value of self.updowngapthreemethods is represented by
self.updowngapthreemethods.Current.Value
            self.Plot("UpDownGapThreeMethods", "updowngapthreemethods",
self.updowngapthreemethods.Current.Value)
```

The following reference table describes the `UpDownGapThreeMethods` method:

INDICATORS

UpDownGapThreeMethods() 1/1

```
UpDownGapThreeMethods QuantConnect.Algorithm.CandlestickPatterns.UpDownGapThreeMethods (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `UpDownGapThreeMethods` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`UpDownGapThreeMethods` - The new `UpDownGapThreeMethods` indicator object.

Definition at line 1015 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `UpDownGapThreeMethods` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class UpDownGapThreeMethodsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.updowngapthreemethods = UpDownGapThreeMethods()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.updowngapthreemethods.Update(bar)

        if self.updowngapthreemethods.IsReady:
            # The current value of self.updowngapthreemethods is represented by
self.updowngapthreemethods.Current.Value
            self.Plot("UpDownGapThreeMethods", "updowngapthreemethods",
self.updowngapthreemethods.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class UpDownGapThreeMethodsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.updowngapthreemethods = UpDownGapThreeMethods()
        self.RegisterIndicator(self.symbol, self.updowngapthreemethods, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.updowngapthreemethods.IsReady:
            # The current value of self.updowngapthreemethods is represented by
self.updowngapthreemethods.Current.Value
            self.Plot("UpDownGapThreeMethods", "updowngapthreemethods",
self.updowngapthreemethods.Current.Value)
```

The following reference table describes the `UpDownGapThreeMethods` constructor:

INDICATORS

UpDownGapThreeMethods() 1/2

```
UpDownGapThreeMethods QuantConnect.Indicators.CandlestickPatterns.UpDownGapThreeMethods (
string name
)
```

Initializes a new instance of the `UpDownGapThreeMethod` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`UpDownGapThreeMethods` - The new `UpDownGapThreeMethods` indicator object.

Definition at line 41 of file Indicators/CandlestickPatterns/UpDownGapThreeMethods.cs.

INDICATORS

UpDownGapThreeMethods() 2/2

```
UpDownGapThreeMethods QuantConnect.Indicators.CandlestickPatterns.UpDownGapThreeMethods (

)
```

Initializes a new instance of the `UpDownGapThreeMethod` class.

Show Details

This method requires no argument input.

Return

`UpDownGapThreeMethods` - The new `UpDownGapThreeMethods` indicator object.

Definition at line 49 of file Indicators/CandlestickPatterns/UpDownGapThreeMethods.cs.

Supported Indicators

Upside Gap Two Crows

Introduction

Create a new Upside Gap Two Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using UpsideGapTwoCrows Indicator

To create an automatic indicators for `UpsideGapTwoCrows` , call the `UpsideGapTwoCrows` helper method from the `QCAlgorithm` class. The `UpsideGapTwoCrows` method creates a `UpsideGapTwoCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class UpsideGapTwoCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.upsidegaptwocrows = self.UpsideGapTwoCrows(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.upsidegaptwocrows.IsReady:
            # The current value of self.upsidegaptwocrows is represented by
self.upsidegaptwocrows.Current.Value
            self.Plot("UpsideGapTwoCrows", "upsidegaptwocrows", self.upsidegaptwocrows.Current.Value)
```

The following reference table describes the `UpsideGapTwoCrows` method:

INDICATORS

UpsideGapTwoCrows() 1/1

```
UpsideGapTwoCrows QuantConnect.Algorithm.CandlestickPatterns.UpsideGapTwoCrows (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new `UpsideGapTwoCrows` pattern indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose pattern we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if null defaults to casting the input value to a TradeBar.

Return

`UpsideGapTwoCrows` - The new `UpsideGapTwoCrows` indicator object.

Definition at line 999 of file Algorithm/CandlestickPatterns.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `UpsideGapTwoCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class UpsideGapTwoCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.upsidegaptwocrows = UpsideGapTwoCrows()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.upsidegaptwocrows.Update(bar)

        if self.upsidegaptwocrows.IsReady:
            # The current value of self.upsidegaptwocrows is represented by
self.upsidegaptwocrows.Current.Value
            self.Plot("UpsideGapTwoCrows", "upsidegaptwocrows", self.upsidegaptwocrows.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
PY
class UpsideGapTwoCrowsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.upsidegaptwocrows = UpsideGapTwoCrows()
        self.RegisterIndicator(self.symbol, self.upsidegaptwocrows, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.upsidegaptwocrows.IsReady:
            # The current value of self.upsidegaptwocrows is represented by
self.upsidegaptwocrows.Current.Value
            self.Plot("UpsideGapTwoCrows", "upsidegaptwocrows", self.upsidegaptwocrows.Current.Value)
```

The following reference table describes the `UpsideGapTwoCrows` constructor:

INDICATORS

UpsideGapTwoCrows() 1/2

```
UpsideGapTwoCrows QuantConnect.Indicators.CandlestickPatterns.UpsideGapTwoCrows (
string name
)
```

Initializes a new instance of the `UpsideGapTwoCrow` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`UpsideGapTwoCrows` - The new `UpsideGapTwoCrows` indicator object.

Definition at line 48 of file Indicators/CandlestickPatterns/UpsideGapTwoCrows.cs.

INDICATORS

UpsideGapTwoCrows() 2/2

```
UpsideGapTwoCrows QuantConnect.Indicators.CandlestickPatterns.UpsideGapTwoCrows (
)
```

Initializes a new instance of the `UpsideGapTwoCrow` class.

Show Details

This method requires no argument input.

Return

`UpsideGapTwoCrows` - The new `UpsideGapTwoCrows` indicator object.

Definition at line 58 of file Indicators/CandlestickPatterns/UpsideGapTwoCrows.cs.

Supported Indicators

Absolute Price Oscillator

Introduction

This indicator computes the Absolute Price Oscillator (APO) The Absolute Price Oscillator is calculated using the following formula: APO[i] = FastMA[i] - SlowMA[i]

To view the implementation of this indicator, see the LEAN GitHub repository .

Using APO Indicator

To create an automatic indicators for `AbsolutePriceOscillator` , call the `APO` helper method from the `QCAlgorithm` class. The `APO` method creates a `AbsolutePriceOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AbsolutePriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.apo = self.APO(self.symbol, 10, 2, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.apo.IsReady:
            # The current value of self.apo is represented by self.apo.Current.Value
            self.Plot("AbsolutePriceOscillator", "apo", self.apo.Current.Value)
            # Plot all attributes of self.apo
            self.Plot("AbsolutePriceOscillator", "fast", self.apo.Fast.Current.Value)
            self.Plot("AbsolutePriceOscillator", "slow", self.apo.Slow.Current.Value)
            self.Plot("AbsolutePriceOscillator", "signal", self.apo.Signal.Current.Value)
            self.Plot("AbsolutePriceOscillator", "histogram", self.apo.Histogram.Current.Value)
```

The following reference table describes the `APO` method:

INDICATORS

APO() 1/1

```
AbsolutePriceOscillator QuantConnect.Algorithm.QCAlgorithm.APO (
Symbol symbol,
Int32 fastPeriod,
Int32 slowPeriod,
MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new AbsolutePriceOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose APO we want.
`Int32`	fastPeriod	The fast moving average period.
`Int32`	slowPeriod	The slow moving average period.
`MovingAverageType`	movingAverageType	The type of moving average to use.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AbsolutePriceOscillator` - The AbsolutePriceOscillator indicator for the requested symbol over the specified period.

Definition at line 205 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `AbsolutePriceOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class AbsolutePriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.apo = AbsolutePriceOscillator(10, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.apo.Update(bar.EndTime, bar.Close)

        if self.apo.IsReady:
            # The current value of self.apo is represented by self.apo.Current.Value
            self.Plot("AbsolutePriceOscillator", "apo", self.apo.Current.Value)
            # Plot all attributes of self.apo
            self.Plot("AbsolutePriceOscillator", "fast", self.apo.Fast.Current.Value)
            self.Plot("AbsolutePriceOscillator", "slow", self.apo.Slow.Current.Value)
            self.Plot("AbsolutePriceOscillator", "signal", self.apo.Signal.Current.Value)
            self.Plot("AbsolutePriceOscillator", "histogram", self.apo.Histogram.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AbsolutePriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.apo = AbsolutePriceOscillator(10, 20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.apo, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.apo.IsReady:
            # The current value of self.apo is represented by self.apo.Current.Value
            self.Plot("AbsolutePriceOscillator", "apo", self.apo.Current.Value)
            # Plot all attributes of self.apo
            self.Plot("AbsolutePriceOscillator", "fast", self.apo.Fast.Current.Value)
            self.Plot("AbsolutePriceOscillator", "slow", self.apo.Slow.Current.Value)
            self.Plot("AbsolutePriceOscillator", "signal", self.apo.Signal.Current.Value)
            self.Plot("AbsolutePriceOscillator", "histogram", self.apo.Histogram.Current.Value)
```

The following reference table describes the `AbsolutePriceOscillator` constructor:

INDICATORS

AbsolutePriceOscillator() 1/2

```
AbsolutePriceOscillator QuantConnect.Indicators.AbsolutePriceOscillator (
string name,
int fastPeriod,
int slowPeriod,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the `AbsolutePriceOscillato` class using the specified name and parameters.

Show Details

Parameters		
string	name	The name of this indicator.
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to use. Default: MovingAverageType.Simple.

Return

AbsolutePriceOscillator - The new AbsolutePriceOscillator indicator object.

Definition at line 35 of file Indicators/AbsolutePriceOscillator.cs.

INDICATORS

AbsolutePriceOscillator() 2/2

```
AbsolutePriceOscillator QuantConnect.Indicators.AbsolutePriceOscillator (
int fastPeriod,
int slowPeriod,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the AbsolutePriceOscillato class using the specified parameters.

Show Details

Parameters		
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to use. Default: MovingAverageType.Simple.

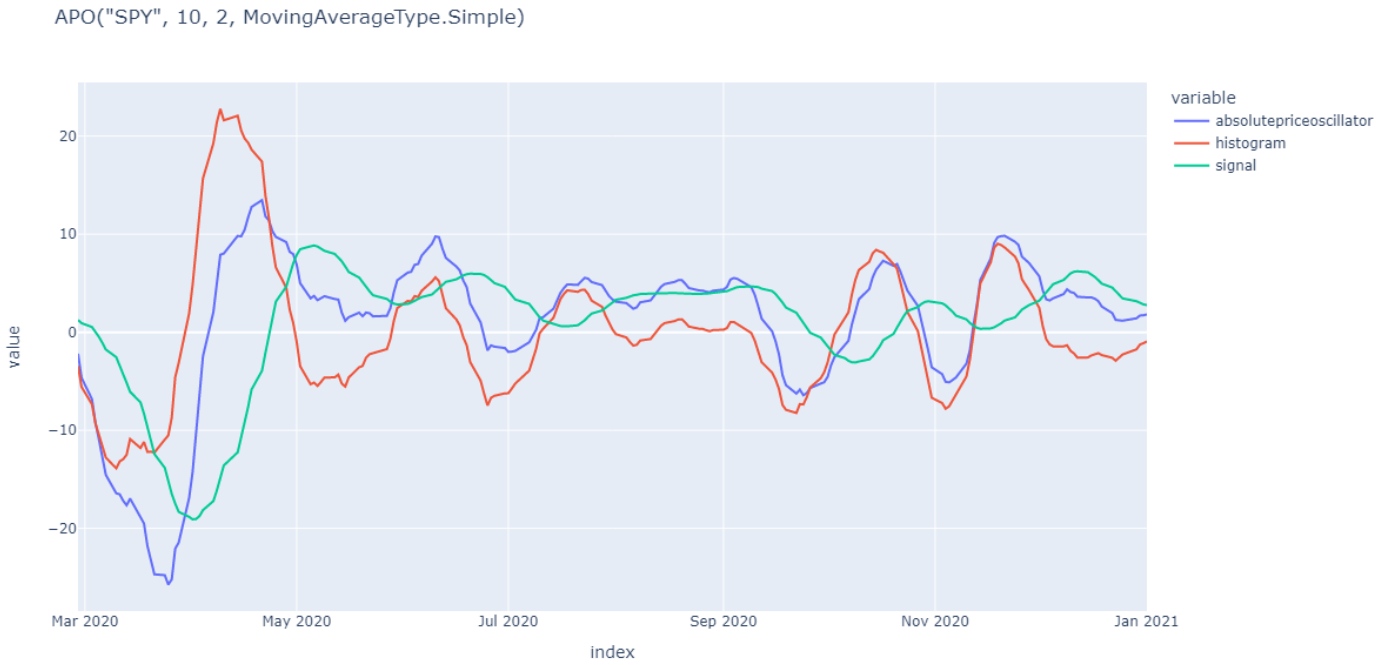
Return

AbsolutePriceOscillator - The new AbsolutePriceOscillator indicator object.

Definition at line 46 of file Indicators/AbsolutePriceOscillator.cs.

Visualization

The following image shows plot values of selected properties of `AbsolutePriceOscillator` using the plotly library.



Supported Indicators

Acceleration Bands

Introduction

The Acceleration Bands created by Price Headley plots upper and lower envelope bands around a moving average.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ABANDS Indicator

To create an automatic indicators for `AccelerationBands` , call the `ABANDS` helper method from the `QCAlgorithm` class. The `ABANDS` method creates a `AccelerationBands` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AccelerationBandsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.abands = self.ABANDS(self.symbol, 10, 4, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.abands.IsReady:
            # The current value of self.abands is represented by self.abands.Current.Value
            self.Plot("AccelerationBands", "abands", self.abands.Current.Value)
            # Plot all attributes of self.abands
            self.Plot("AccelerationBands", "middleband", self.abands.MiddleBand.Current.Value)
            self.Plot("AccelerationBands", "upperband", self.abands.UpperBand.Current.Value)
            self.Plot("AccelerationBands", "lowerband", self.abands.LowerBand.Current.Value)
```

The following reference table describes the `ABANDS` method:

INDICATORS

ABANDS() 1/1

```
AccelerationBands QuantConnect.Algorithm.QCAlgorithm.ABANDS (
Symbol symbol,
Int32 period,
*Decimal width,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Acceleration Bands indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Acceleration Bands we want.
`Int32`	period	The period of the three moving average (middle, upper and lower band).
`*Decimal`	width	*(Optional)* A coefficient specifying the distance between the middle band and upper or lower bands.
`*MovingAverageType`	movingAverageType	*(Optional)* Type of the moving average. *Options: ['Simple', 'Exponential', 'Wilders', 'LinearWeightedMovingAverage', 'DoubleExponential', 'TripleExponential', 'Triangular', 'T3', 'Kama', 'Hull', 'Alma']*
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AccelerationBands` - The new `AccelerationBands` object.

Definition at line 46 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `AccelerationBands` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class AccelerationBandsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.abands = AccelerationBands("", 10, 4, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.abands.Update(bar.EndTime, bar.Close)

        if self.abands.IsReady:
            # The current value of self.abands is represented by self.abands.Current.Value
            self.Plot("AccelerationBands", "abands", self.abands.Current.Value)
            # Plot all attributes of self.abands
            self.Plot("AccelerationBands", "middleband", self.abands.MiddleBand.Current.Value)
            self.Plot("AccelerationBands", "upperband", self.abands.UpperBand.Current.Value)
            self.Plot("AccelerationBands", "lowerband", self.abands.LowerBand.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AccelerationBandsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.abands = AccelerationBands("", 10, 4, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.abands, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.abands.IsReady:
            # The current value of self.abands is represented by self.abands.Current.Value
            self.Plot("AccelerationBands", "abands", self.abands.Current.Value)
            # Plot all attributes of self.abands
            self.Plot("AccelerationBands", "middleband", self.abands.MiddleBand.Current.Value)
            self.Plot("AccelerationBands", "upperband", self.abands.UpperBand.Current.Value)
            self.Plot("AccelerationBands", "lowerband", self.abands.LowerBand.Current.Value)
```

The following reference table describes the `AccelerationBands` constructor:

INDICATORS

AccelerationBands() 1/3

```
AccelerationBands QuantConnect.Indicators.AccelerationBands (
string name,
int period,
decimal width,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the `AccelerationBands` class.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the three moving average (middle, upper and lower band).
decimal	width	A coefficient specifying the distance between the middle band and upper or lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* Type of the moving average. Default: MovingAverageType.Simple.

Return

`AccelerationBands` - The new `AccelerationBands` indicator object.

Definition at line 55 of file Indicators/AccelerationBands.cs.

INDICATORS

AccelerationBands() 2/3

```
AccelerationBands QuantConnect.Indicators.AccelerationBands (
int period,
decimal width
)
```

Initializes a new instance of the `AccelerationBands` class.

Show Details

Parameters		
int	period	The period of the three moving average (middle, upper and lower band).
decimal	width	A coefficient specifying the distance between the middle band and upper or lower bands.

Return

`AccelerationBands` - The new `AccelerationBands` indicator object.

Definition at line 72 of file Indicators/AccelerationBands.cs.

INDICATORS

AccelerationBands() 3/3

```
AccelerationBands QuantConnect.Indicators.AccelerationBands (
int period
)
```

Initializes a new instance of the `AccelerationBands` class.

Show Details

Parameters		
`int`	period	The period of the three moving average (middle, upper and lower band).

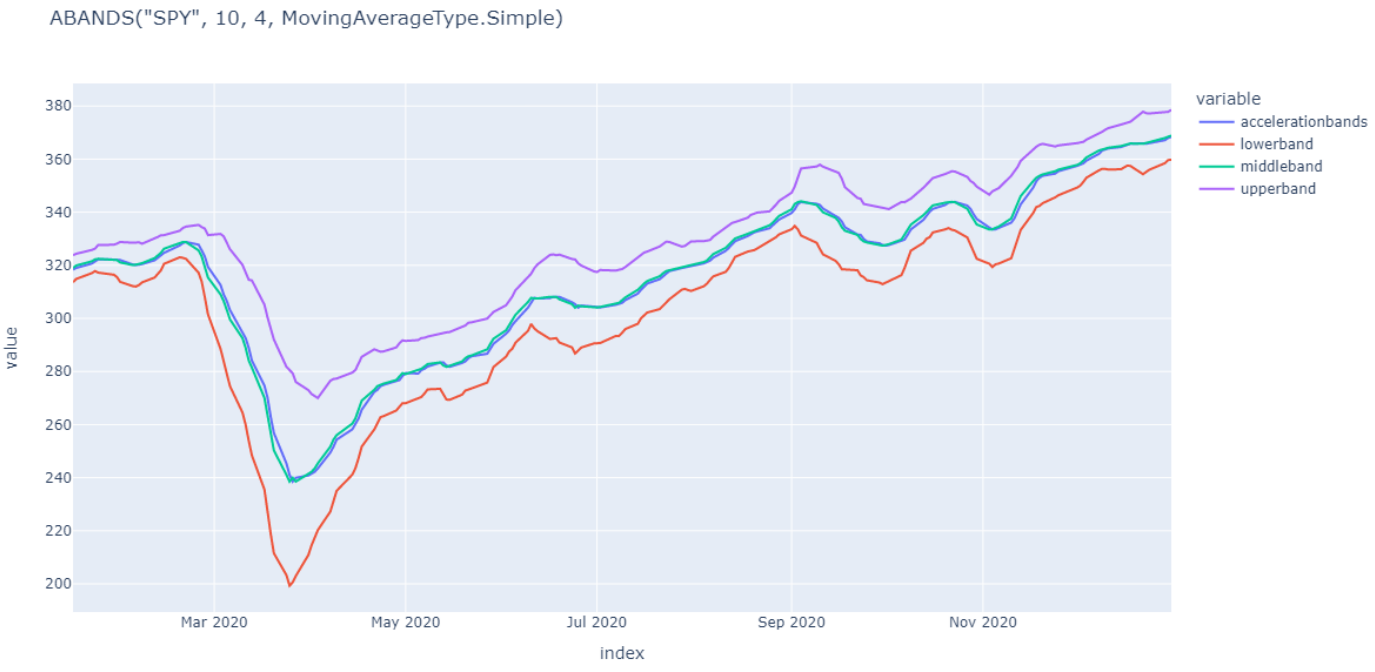
Return

`AccelerationBands` - The new `AccelerationBands` indicator object.

Definition at line 81 of file Indicators/AccelerationBands.cs.

Visualization

The following image shows plot values of selected properties of `AccelerationBands` using the plotly library.



Supported Indicators

Accumulation Distribution

Introduction

This indicator computes the Accumulation/Distribution (AD) The Accumulation/Distribution is calculated using the following formula: AD = AD + ((Close - Low) - (High - Close)) / (High - Low) * Volume

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AD Indicator

To create an automatic indicators for `AccumulationDistribution` , call the `AD` helper method from the `QCAlgorithm` class. The `AD` method creates a `AccumulationDistribution` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class AccumulationDistributionAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ad = self.AD(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.ad.IsReady:
            # The current value of self.ad is represented by self.ad.Current.Value
            self.Plot("AccumulationDistribution", "ad", self.ad.Current.Value)
```

The following reference table describes the `AD` method:

INDICATORS

AD() 1/1

```
AccumulationDistribution QuantConnect.Algorithm.QCAlgorithm.AD (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new AccumulationDistribution indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose AD we want.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AccumulationDistribution` - The AccumulationDistribution indicator for the requested symbol over the specified period.

Definition at line 64 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AccumulationDistribution` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```
class AccumulationDistributionAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ad = AccumulationDistribution()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ad.Update(bar)

        if self.ad.IsReady:
            # The current value of self.ad is represented by self.ad.Current.Value
            self.Plot("AccumulationDistribution", "ad", self.ad.Current.Value)
```

PY

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class AccumulationDistributionAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ad = AccumulationDistribution()
        self.RegisterIndicator(self.symbol, self.ad, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ad.IsReady:
            # The current value of self.ad is represented by self.ad.Current.Value
            self.Plot("AccumulationDistribution", "ad", self.ad.Current.Value)
```

The following reference table describes the `AccumulationDistribution` constructor:

INDICATORS

AccumulationDistribution() 1/2

```
AccumulationDistribution QuantConnect.Indicators.AccumulationDistribution (

)
```

Initializes a new instance of the `AccumulationDistributio` class using the specified name.

Show Details

This method requires no argument input.

Return

`AccumulationDistribution` - The new `AccumulationDistribution` indicator object.

Definition at line 30 of file Indicators/AccumulationDistribution.cs.

INDICATORS

AccumulationDistribution() 2/2

```
AccumulationDistribution QuantConnect.Indicators.AccumulationDistribution (
string name
)
```

Initializes a new instance of the `AccumulationDistributio` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`AccumulationDistribution` - The new `AccumulationDistribution` indicator object.

Definition at line 39 of file Indicators/AccumulationDistribution.cs.

Visualization

The following image shows plot values of selected properties of `AccumulationDistribution` using the plotly library.

Supported Indicators

Accumulation Distribution Oscillator

Introduction

This indicator computes the Accumulation/Distribution Oscillator (ADOSC) The Accumulation/Distribution Oscillator is calculated using the following formula: ADOSC = EMA(fast,AD) - EMA(slow,AD)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADOSC Indicator

To create an automatic indicators for `AccumulationDistributionOscillator` , call the `ADOSC` helper method from the `QCAlgorithm` class. The `ADOSC` method creates a `AccumulationDistributionOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AccumulationDistributionOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adosc = self.ADOSC(self.symbol, 10, 2)

    def OnData(self, slice: Slice) -> None:
        if self.adosc.IsReady:
            # The current value of self.adosc is represented by self.adosc.Current.Value
            self.Plot("AccumulationDistributionOscillator", "adosc", self.adosc.Current.Value)
```

The following reference table describes the `ADOSC` method:

INDICATORS

ADOSC() 1/1

```
AccumulationDistributionOscillator QuantConnect.Algorithm.QCAlgorithm.ADOSC (
Symbol symbol,
Int32 fastPeriod,
Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new AccumulationDistributionOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ADOSC we want.
`Int32`	fastPeriod	The fast moving average period.
`Int32`	slowPeriod	The slow moving average period.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AccumulationDistributionOscillator` - The AccumulationDistributionOscillator indicator for the requested symbol over the specified period.

Definition at line 83 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AccumulationDistributionOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class AccumulationDistributionOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adosc = AccumulationDistributionOscillator(10, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.adosc.Update(bar)

        if self.adosc.IsReady:
            # The current value of self.adosc is represented by self.adosc.Current.Value
            self.Plot("AccumulationDistributionOscillator", "adosc", self.adosc.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
PY
class AccumulationDistributionOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adosc = AccumulationDistributionOscillator(10, 20)
        self.RegisterIndicator(self.symbol, self.adosc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.adosc.IsReady:
            # The current value of self.adosc is represented by self.adosc.Current.Value
            self.Plot("AccumulationDistributionOscillator", "adosc", self.adosc.Current.Value)
```

The following reference table describes the `AccumulationDistributionOscillator` constructor:

INDICATORS

AccumulationDistributionOscillator() 1/2

```
AccumulationDistributionOscillator QuantConnect.Indicators.AccumulationDistributionOscillator (
int fastPeriod,
int slowPeriod
)
```

Initializes a new instance of the `AccumulationDistributionOscillato` class using the specified parameters.

Show Details

Parameters		
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.

Return

`AccumulationDistributionOscillator` - The new `AccumulationDistributionOscillator` indicator object.

Definition at line 38 of file Indicators/AccumulationDistributionOscillator.cs.

INDICATORS

AccumulationDistributionOscillator() 2/2

```
AccumulationDistributionOscillator QuantConnect.Indicators.AccumulationDistributionOscillator (
string name,
int fastPeriod,
int slowPeriod
)
```

Initializes a new instance of the `AccumulationDistributionOscillato` class using the specified parameters.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	fastPeriod	The fast moving average period.
`int`	slowPeriod	The slow moving average period.

Return

`AccumulationDistributionOscillator` - The new `AccumulationDistributionOscillator` indicator object.

Definition at line 49 of file Indicators/AccumulationDistributionOscillator.cs.

Visualization

The following image shows plot values of selected properties of `AccumulationDistributionOscillator` using the plotly library.

ADOSC("SPY", 10, 2)

Supported Indicators

Advance Decline Difference

Introduction

The Advance Decline Difference compute the difference between the number of stocks that closed higher and the number of stocks that closed lower than their previous day's closing prices.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADDIFF Indicator

To create an automatic indicators for `AdvanceDeclineDifference` , call the `ADDIFF` helper method from the `QCAlgorithm` class. The `ADDIFF` method creates a `AdvanceDeclineDifference` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class AdvanceDeclineDifferenceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.addiff = self.ADDIFF([self.symbol, Symbol.Create("QQQ", SecurityType.Equity, Market.USA)])

    def OnData(self, slice: Slice) -> None:
        if self.addiff.IsReady:
            # The current value of self.addiff is represented by self.addiff.Current.Value
            self.Plot("AdvanceDeclineDifference", "addiff", self.addiff.Current.Value)
```

The following reference table describes the `ADDIFF` method:

INDICATORS

ADDIFF() 1/1

```
AdvanceDeclineDifference QuantConnect.Algorithm.QCAlgorithm.ADDIFF (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* /

Return

`AdvanceDeclineDifference` - The new `AdvanceDeclineDifference` object.

Definition at line 2021 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AdvanceDeclineDifference` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```
PY
class AdvanceDeclineDifferenceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.addiff = AdvanceDeclineDifference("")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.addiff.Update(bar)

        if self.addiff.IsReady:
            # The current value of self.addiff is represented by self.addiff.Current.Value
            self.Plot("AdvanceDeclineDifference", "addiff", self.addiff.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AdvanceDeclineDifferenceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.addiff = AdvanceDeclineDifference("")
        self.RegisterIndicator(self.symbol, self.addiff, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.addiff.IsReady:
            # The current value of self.addiff is represented by self.addiff.Current.Value
            self.Plot("AdvanceDeclineDifference", "addiff", self.addiff.Current.Value)
```

The following reference table describes the `AdvanceDeclineDifference` constructor:

INDICATORS

AdvanceDeclineDifference() 1/1

```
AdvanceDeclineDifference QuantConnect.Indicators.AdvanceDeclineDifference (
)
```

Initializes a new instance of the `AdvanceDeclineDifferenc` class.

Show Details

This method requires no argument input.

Return

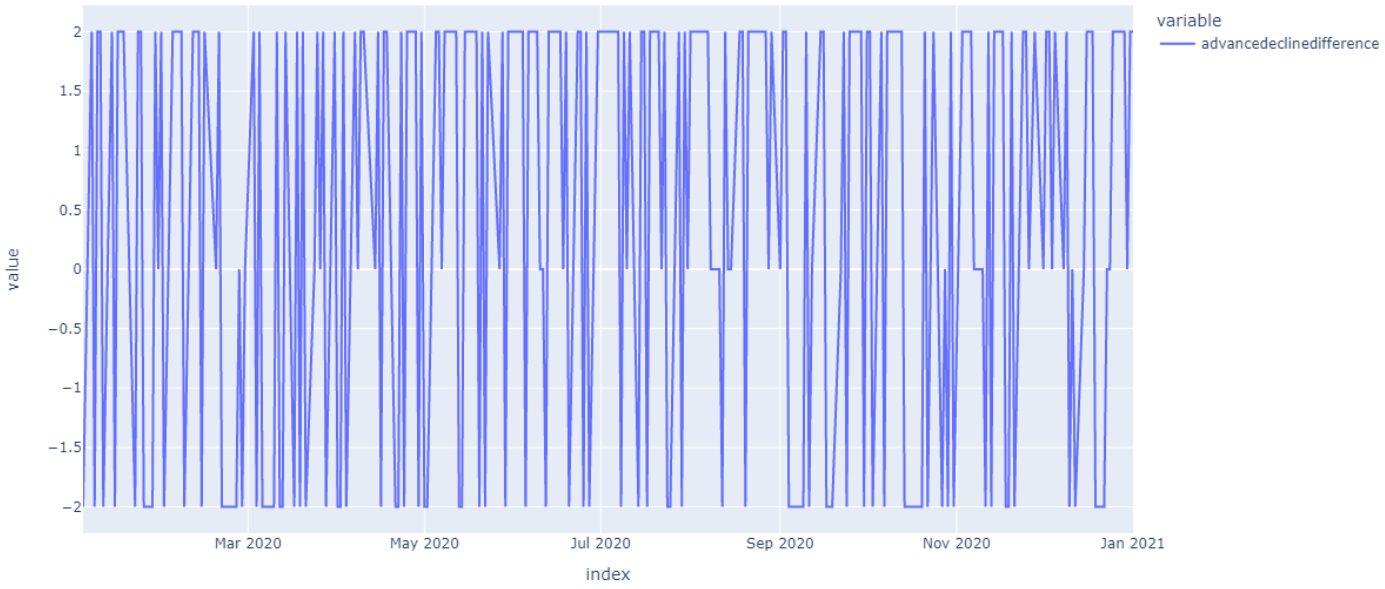
`AdvanceDeclineDifference` - The new `AdvanceDeclineDifference` indicator object.

Definition at line 29 of file Indicators/AdvanceDeclineDifference.cs.

Visualization

The following image shows plot values of selected properties of `AdvanceDeclineDifference` using the plotly library.

ADDIFF(["SPY", "QQQ"])



Supported Indicators

Advance Decline Ratio

Introduction

The advance-decline ratio (ADR) compares the number of stocks that closed higher against the number of stocks that closed lower than their previous day's closing prices.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADR Indicator

To create an automatic indicators for `AdvanceDeclineRatio` , call the `ADR` helper method from the `QCAlgorithm` class. The `ADR` method creates a `AdvanceDeclineRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class AdvanceDeclineRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adr = self.ADR([self.symbol, Symbol.Create("QQQ", SecurityType.Equity, Market.USA)])

    def OnData(self, slice: Slice) -> None:
        if self.adr.IsReady:
            # The current value of self.adr is represented by self.adr.Current.Value
            self.Plot("AdvanceDeclineRatio", "adr", self.adr.Current.Value)
```

The following reference table describes the `ADR` method:

INDICATORS

ADR() 1/1

```
AdvanceDeclineRatio QuantConnect.Algorithm.QCAlgorithm.ADR (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* /

Return

`AdvanceDeclineRatio` - The new `AdvanceDeclineRatio` object.

Definition at line 1979 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AdvanceDeclineRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class AdvanceDeclineRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adr = AdvanceDeclineRatio("")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.adr.Update(bar)

        if self.adr.IsReady:
            # The current value of self.adr is represented by self.adr.Current.Value
            self.Plot("AdvanceDeclineRatio", "adr", self.adr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AdvanceDeclineRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adr = AdvanceDeclineRatio("")
        self.RegisterIndicator(self.symbol, self.adr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.adr.IsReady:
            # The current value of self.adr is represented by self.adr.Current.Value
            self.Plot("AdvanceDeclineRatio", "adr", self.adr.Current.Value)
```

The following reference table describes the `AdvanceDeclineRatio` constructor:

INDICATORS

AdvanceDeclineRatio() 1/1

```
AdvanceDeclineRatio QuantConnect.Indicators.AdvanceDeclineRatio (

)
```

Initializes a new instance of the `AdvanceDeclineRati` class.

Show Details

This method requires no argument input.

Return

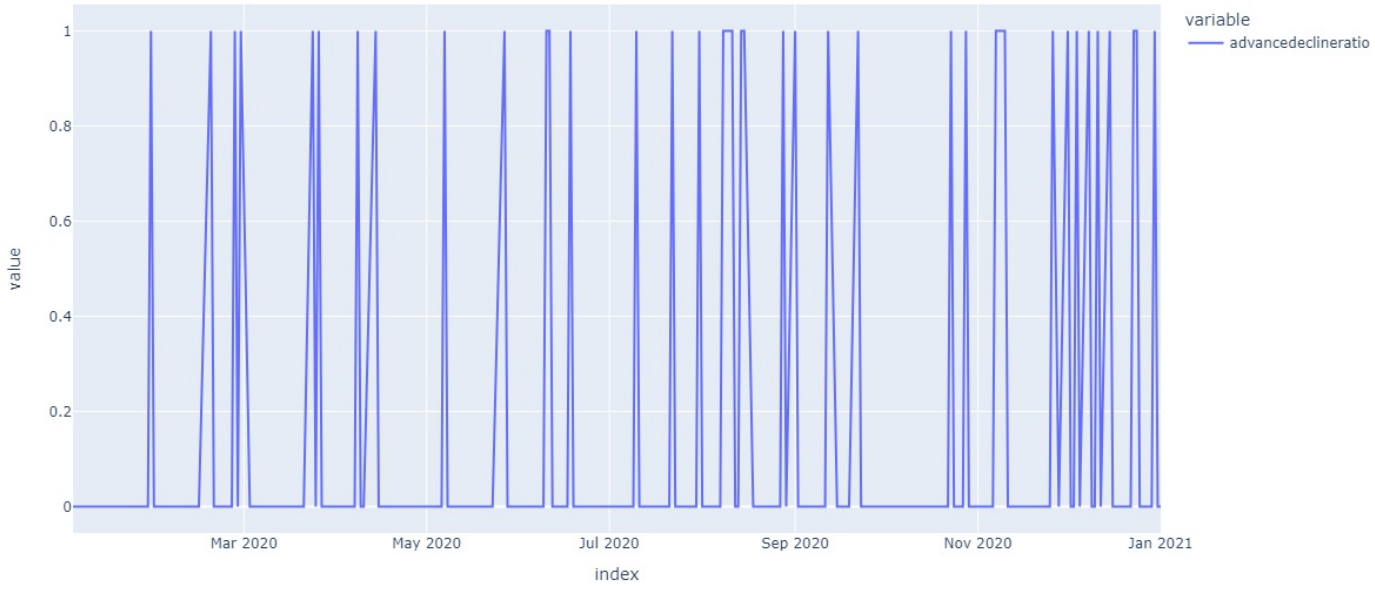
`AdvanceDeclineRatio` - The new `AdvanceDeclineRatio` indicator object.

Definition at line 30 of file Indicators/AdvanceDeclineRatio.cs.

Visualization

The following image shows plot values of selected properties of `AdvanceDeclineRatio` using the plotly library.

ADR(["SPY", "QQQ"])


variable
advancedeclineratio
value
1
0.8
0.6
0.4
0.2
0
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Advance Decline Volume Ratio

Introduction

The Advance Decline Volume Ratio is a Breadth indicator calculated as ratio of summary volume of advancing stocks to summary volume of declining stocks. AD Volume Ratio is used in technical analysis to see where the main trading activity is focused.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADVR Indicator

To create an automatic indicators for `AdvanceDeclineVolumeRatio` , call the `ADVR` helper method from the `QCAlgorithm` class. The `ADVR` method creates a `AdvanceDeclineVolumeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AdvanceDeclineVolumeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.advr = self.ADVR([self.symbol, Symbol.Create("QQQ", SecurityType.Equity, Market.USA)])

    def OnData(self, slice: Slice) -> None:
        if self.advr.IsReady:
            # The current value of self.advr is represented by self.advr.Current.Value
            self.Plot("AdvanceDeclineVolumeRatio", "advr", self.advr.Current.Value)
```

The following reference table describes the `ADVR` method:

INDICATORS

ADVR() 1/1

```
AdvanceDeclineVolumeRatio QuantConnect.Algorithm.QCAlgorithm.ADVR (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* /

Return

`AdvanceDeclineVolumeRatio` - The new `AdvanceDeclineVolumeRatio` object.

Definition at line 2000 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AdvanceDeclineVolumeRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```
PY
class AdvanceDeclineVolumeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.advr = AdvanceDeclineVolumeRatio("")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.advr.Update(bar)

        if self.advr.IsReady:
            # The current value of self.advr is represented by self.advr.Current.Value
            self.Plot("AdvanceDeclineVolumeRatio", "advr", self.advr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AdvanceDeclineVolumeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.advr = AdvanceDeclineVolumeRatio("")
        self.RegisterIndicator(self.symbol, self.advr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.advr.IsReady:
            # The current value of self.advr is represented by self.advr.Current.Value
            self.Plot("AdvanceDeclineVolumeRatio", "advr", self.advr.Current.Value)
```

The following reference table describes the `AdvanceDeclineVolumeRatio` constructor:

INDICATORS

AdvanceDeclineVolumeRatio() 1/1

```
AdvanceDeclineVolumeRatio QuantConnect.Indicators.AdvanceDeclineVolumeRatio (

)
```

Initializes a new instance of the `AdvanceDeclineVolumeRati` class.

Show Details

This method requires no argument input.

Return

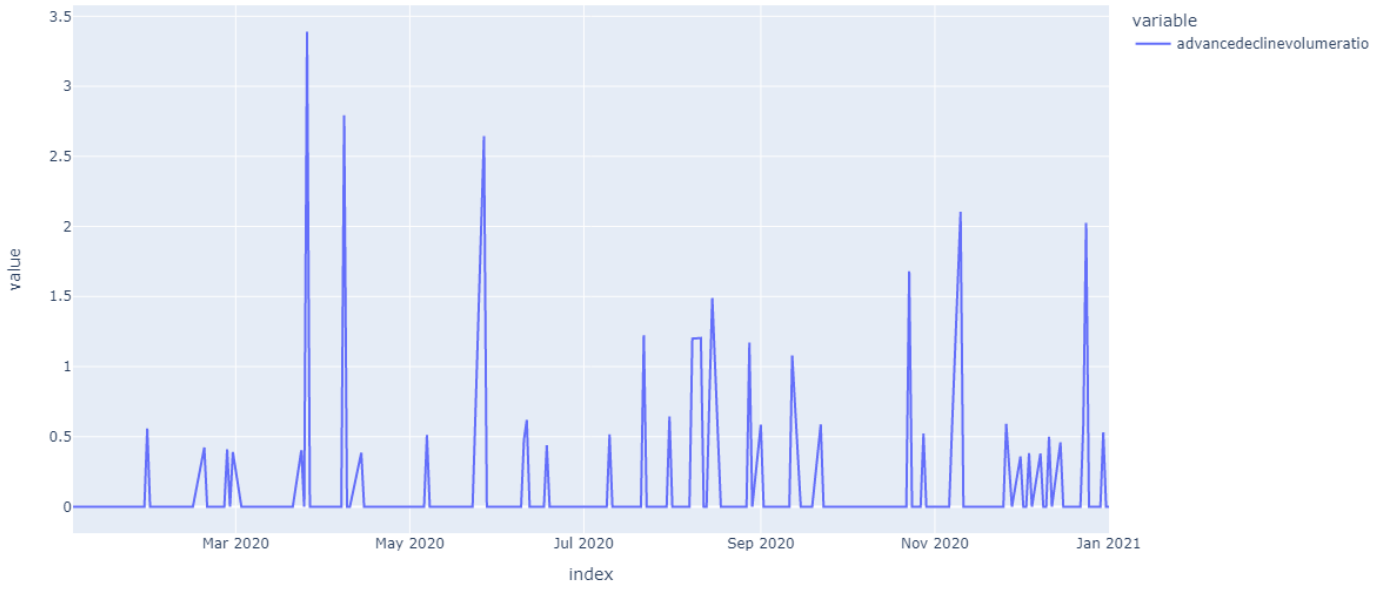
`AdvanceDeclineVolumeRatio` - The new `AdvanceDeclineVolumeRatio` indicator object.

Definition at line 30 of file Indicators/AdvanceDeclineVolumeRatio.cs.

Visualization

The following image shows plot values of selected properties of `AdvanceDeclineVolumeRatio` using the plotly library.

ADVR(["SPY", "QQQ"])


variable
advancedeclinevolumeratio
value
index
3.5
3
2.5
2
1.5
1
0.5
0
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021

Supported Indicators

Arms Index

Introduction

The Arms Index, also called the Short-Term Trading Index (TRIN) is a technical analysis indicator that compares the number of advancing and declining stocks (AD Ratio) to advancing and declining volume (AD volume).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TRIN Indicator

To create an automatic indicators for `ArmsIndex` , call the `TRIN` helper method from the `QCAlgorithm` class. The `TRIN` method creates a `ArmsIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ArmsIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trin = self.TRIN([self.symbol, Symbol.Create("QQQ", SecurityType.Equity, Market.USA)])

    def OnData(self, slice: Slice) -> None:
        if self.trin.IsReady:
            # The current value of self.trin is represented by self.trin.Current.Value
            self.Plot("ArmsIndex", "trin", self.trin.Current.Value)
            # Plot all attributes of self.trin
            self.Plot("ArmsIndex", "adratio", self.trin.ADRatio.Current.Value)
            self.Plot("ArmsIndex", "advratio", self.trin.ADVRatio.Current.Value)
```

The following reference table describes the `TRIN` method:

INDICATORS

TRIN() 1/2

```
ArmsIndex QuantConnect.Algorithm.QCAlgorithm.TRIN (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Arms Index indicator.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, TradeBar>	selector	*(Optional)* /

Return

ArmsIndex - The new ArmsIndex object.

Definition at line 1945 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TRIN() 2/2

```
ArmsIndex QuantConnect.Algorithm.QCAlgorithm.TRIN (
Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Arms Index indicator.

Show Details

Parameters		
Symbol>	symbols	The symbols whose Arms Index we want.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, TradeBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

ArmsIndex - The Arms Index indicator for the requested symbol over the specified period.

Definition at line 1958 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ArmsIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class ArmsIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trin = ArmsIndex("")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.trin.Update(bar)

        if self.trin.IsReady:
            # The current value of self.trin is represented by self.trin.Current.Value
            self.Plot("ArmsIndex", "trin", self.trin.Current.Value)
            # Plot all attributes of self.trin
            self.Plot("ArmsIndex", "adratio", self.trin.ADRatio.Current.Value)
            self.Plot("ArmsIndex", "advratio", self.trin.ADVRatio.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class ArmsIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trin = ArmsIndex("")
        self.RegisterIndicator(self.symbol, self.trin, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.trin.IsReady:
            # The current value of self.trin is represented by self.trin.Current.Value
            self.Plot("ArmsIndex", "trin", self.trin.Current.Value)
            # Plot all attributes of self.trin
            self.Plot("ArmsIndex", "adratio", self.trin.ADRatio.Current.Value)
            self.Plot("ArmsIndex", "advratio", self.trin.ADVRatio.Current.Value)
```

The following reference table describes the `ArmsIndex` constructor:

INDICATORS

ArmsIndex() 1/1

```
ArmsIndex QuantConnect.Indicators.ArmsIndex (
)
```

Initializes a new instance of the `ArmsInde` class.

Show Details

This method requires no argument input.

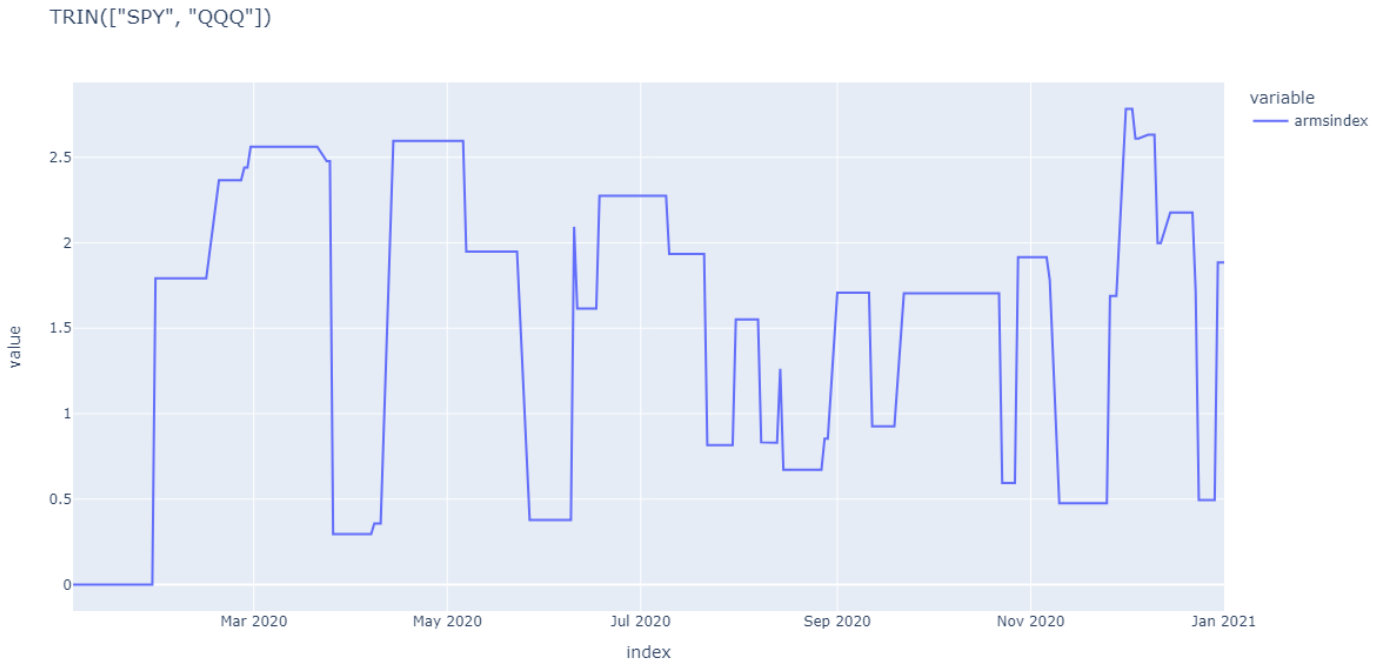
Return

`ArmsIndex` - The new `ArmsIndex` indicator object.

Definition at line 43 of file Indicators/ArmsIndex.cs.

Visualization

The following image shows plot values of selected properties of `ArmsIndex` using the plotly library.



Supported Indicators

Arnaud Legoux Moving Average

Introduction

Smooth and high sensitive moving Average. This moving average reduce lag of the information but still being smooth to reduce noises. Is a weighted moving average, which weights have a Normal shape; the parameters Sigma and Offset affect the kurtosis and skewness of the weights respectively. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ALMA Indicator

To create an automatic indicators for `ArnaudLegouxMovingAverage` , call the `ALMA` helper method from the `QCAlgorithm` class. The `ALMA` method creates a `ArnaudLegouxMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ArnaudLegouxMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.alma = self.ALMA(self.symbol, 10, 6, 0.85)

    def OnData(self, slice: Slice) -> None:
        if self.alma.IsReady:
            # The current value of self.alma is represented by self.alma.Current.Value
            self.Plot("ArnaudLegouxMovingAverage", "alma", self.alma.Current.Value)
```

The following reference table describes the `ALMA` method:

INDICATORS

ALMA() 1/1

```
ArnaudLegouxMovingAverage QuantConnect.Algorithm.QCAlgorithm.ALMA (
Symbol symbol,
Int32 period,
*Int32 sigma,
*Decimal offset,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new ArnaudLegouxMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ALMA we want.
`Int32`	period	int - the number of periods to calculate the ALMA.
`*Int32`	sigma	*(Optional)* int - this parameter is responsible for the shape of the curve coefficients. .
`*Decimal`	offset	*(Optional)* decimal - This parameter allows regulating the smoothness and high sensitivity of the Moving Average. The range for this parameter is [0, 1]. .
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ArnaudLegouxMovingAverage` - The ArnaudLegouxMovingAverage indicator for the requested symbol over the specified period.

Definition at line 185 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ArnaudLegouxMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class ArnaudLegouxMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.alma = ArnaudLegouxMovingAverage(10, 6, 0.85)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.alma.Update(bar.EndTime, bar.Close)

        if self.alma.IsReady:
            # The current value of self.alma is represented by self.alma.Current.Value
            self.Plot("ArnaudLegouxMovingAverage", "alma", self.alma.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class ArnaudLegouxMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.alma = ArnaudLegouxMovingAverage(10, 6, 0.85)
        self.RegisterIndicator(self.symbol, self.alma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.alma.IsReady:
            # The current value of self.alma is represented by self.alma.Current.Value
            self.Plot("ArnaudLegouxMovingAverage", "alma", self.alma.Current.Value)
```

The following reference table describes the `ArnaudLegouxMovingAverage` constructor:

INDICATORS

ArnaudLegouxMovingAverage() 1/4

```
ArnaudLegouxMovingAverage QuantConnect.Indicators.ArnaudLegouxMovingAverage (
string name,
int period
)
```

Initializes a new instance of the `ArnaudLegouxMovingAverage` class.

Show Details

Parameters		
`string`	name	string - a name for the indicator.
`int`	period	int - the number of periods to calculate the ALMA.

Return

`ArnaudLegouxMovingAverage` - The new `ArnaudLegouxMovingAverage` indicator object.

Definition at line 50 of file Indicators/ArnaudLegouxMovingAverage.cs.

INDICATORS

ArnaudLegouxMovingAverage() 2/4

```
ArnaudLegouxMovingAverage QuantConnect.Indicators.ArnaudLegouxMovingAverage (
string name,
int period
)
```

Initializes a new instance of the `ArnaudLegouxMovingAverage` class.

Show Details

Parameters		
`string`	name	string - a name for the indicator.
`int`	period	int - the number of periods to calculate the ALMA.

Return

`ArnaudLegouxMovingAverage` - The new `ArnaudLegouxMovingAverage` indicator object.

Definition at line 72 of file Indicators/ArnaudLegouxMovingAverage.cs.

INDICATORS

ArnaudLegouxMovingAverage() 3/4

```
ArnaudLegouxMovingAverage QuantConnect.Indicators.ArnaudLegouxMovingAverage (
int period
)
```

Initializes a new instance of the `ArnaudLegouxMovingAverage` class.

Show Details

Parameters		
int	period	int - the number of periods to calculate the ALMA.

Return

`ArnaudLegouxMovingAverage` - The new `ArnaudLegouxMovingAverage` indicator object.

Definition at line 89 of file Indicators/ArnaudLegouxMovingAverage.cs.

INDICATORS

ArnaudLegouxMovingAverage() 4/4

```
ArnaudLegouxMovingAverage QuantConnect.Indicators.ArnaudLegouxMovingAverage (
int period
)
```

Initializes a new instance of the `ArnaudLegouxMovingAverage` class.

Show Details

Parameters		
int	period	int - the number of periods to calculate the ALMA.

Return

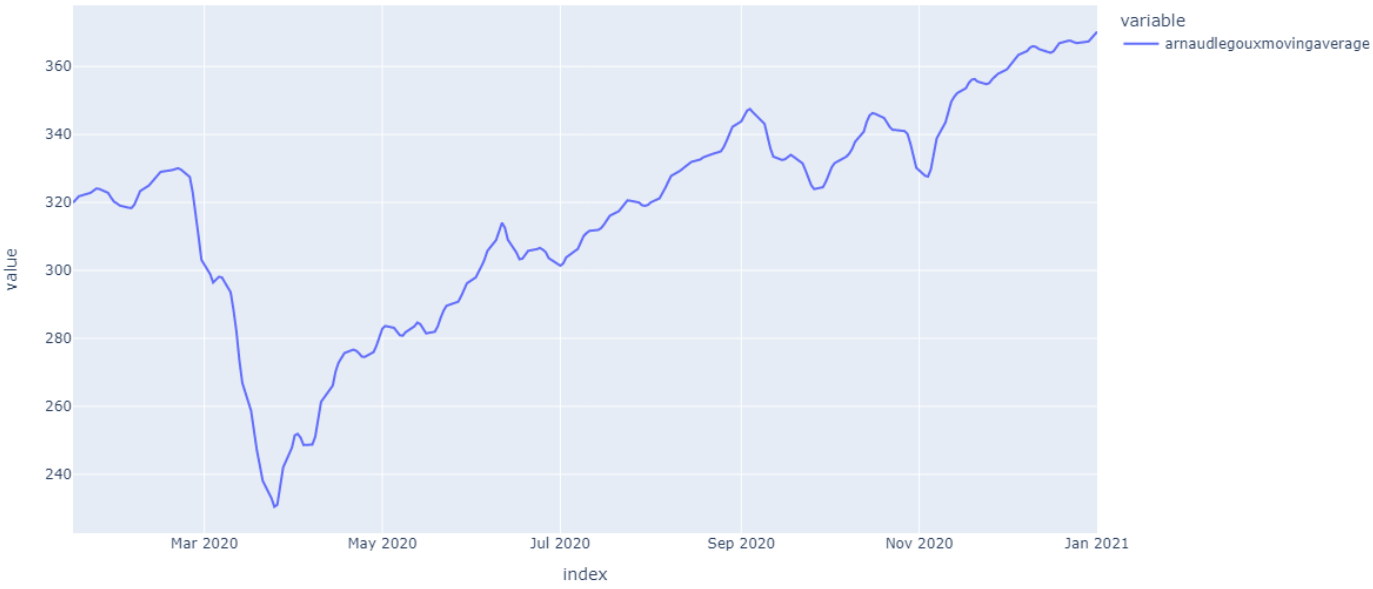
`ArnaudLegouxMovingAverage` - The new `ArnaudLegouxMovingAverage` indicator object.

Definition at line 98 of file Indicators/ArnaudLegouxMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `ArnaudLegouxMovingAverage` using the plotly library.

ALMA("SPY", 10, 6, 0.85)



Supported Indicators

Aroon Oscillator

Introduction

The Aroon Oscillator is the difference between AroonUp and AroonDown. The value of this indicator fluctuates between -100 and +100. An upward trend bias is present when the oscillator is positive, and a negative trend bias is present when the oscillator is negative. AroonUp/Down values over 75 identify strong trends in their respective direction.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AROON Indicator

To create an automatic indicators for `AroonOscillator` , call the `AROON` helper method from the `QCAlgorithm` class. The `AROON` method creates a `AroonOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AroonOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.aroon = self.AROON(self.symbol, 10, 20)

    def OnData(self, slice: Slice) -> None:
        if self.aroon.IsReady:
            # The current value of self.aroon is represented by self.aroon.Current.Value
            self.Plot("AroonOscillator", "aroon", self.aroon.Current.Value)
            # Plot all attributes of self.aroon
            self.Plot("AroonOscillator", "aroonup", self.aroon.AroonUp.Current.Value)
            self.Plot("AroonOscillator", "aroondown", self.aroon.AroonDown.Current.Value)
```

The following reference table describes the `AROON` method:

INDICATORS

AROON() 1/2

```
AroonOscillator QuantConnect.Algorithm.QCAlgorithm.AROON (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AroonOscillator indicator which will compute the AroonUp and AroonDown (as well as the delta).

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Aroon we seek.
`Int32`	period	The look back period for computing number of periods since maximum and minimum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AroonOscillator` - An AroonOscillator configured with the specified periods.

Definition at line 223 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

AROON() 2/2

```
AroonOscillator QuantConnect.Algorithm.QCAlgorithm.AROON (
 Symbol symbol,
 Int32 upPeriod,
 Int32 downPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AroonOscillator indicator which will compute the AroonUp and AroonDown (as well as the delta).

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Aroon we seek.
`Int32`	upPeriod	The look back period for computing number of periods since maximum.
`Int32`	downPeriod	The look back period for computing number of periods since minimum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AroonOscillator` - An AroonOscillator configured with the specified periods.

Definition at line 238 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AroonOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class AroonOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.aroon = AroonOscillator(10, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.aroon.Update(bar)

        if self.aroon.IsReady:
            # The current value of self.aroon is represented by self.aroon.Current.Value
            self.Plot("AroonOscillator", "aroon", self.aroon.Current.Value)
            # Plot all attributes of self.aroon
            self.Plot("AroonOscillator", "aroonup", self.aroon.AroonUp.Current.Value)
            self.Plot("AroonOscillator", "aroondown", self.aroon.AroonDown.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AroonOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.aroon = AroonOscillator(10, 20)
        self.RegisterIndicator(self.symbol, self.aroon, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.aroon.IsReady:
            # The current value of self.aroon is represented by self.aroon.Current.Value
            self.Plot("AroonOscillator", "aroon", self.aroon.Current.Value)
            # Plot all attributes of self.aroon
            self.Plot("AroonOscillator", "aroonup", self.aroon.AroonUp.Current.Value)
            self.Plot("AroonOscillator", "aroondown", self.aroon.AroonDown.Current.Value)
```

The following reference table describes the `AroonOscillator` constructor:

INDICATORS

AroonOscillator() 1/2

```
AroonOscillator QuantConnect.Indicators.AroonOscillator (
int upPeriod,
int downPeriod
)
```

Creates a new AroonOscillator from the specified up/down periods.

Show Details

Parameters		
int	upPeriod	The lookback period to determine the highest high for the AroonDown.
int	downPeriod	The lookback period to determine the lowest low for the AroonUp.

Return

`AroonOscillator` - The new `AroonOscillator` indicator object.

Definition at line 54 of file Indicators/AroonOscillator.cs.

INDICATORS

AroonOscillator() 2/2

```
AroonOscillator QuantConnect.Indicators.AroonOscillator (
string name,
int upPeriod,
int downPeriod
)
```

Creates a new AroonOscillator from the specified up/down periods.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	upPeriod	The lookback period to determine the highest high for the AroonDown.
`int`	downPeriod	The lookback period to determine the lowest low for the AroonUp.

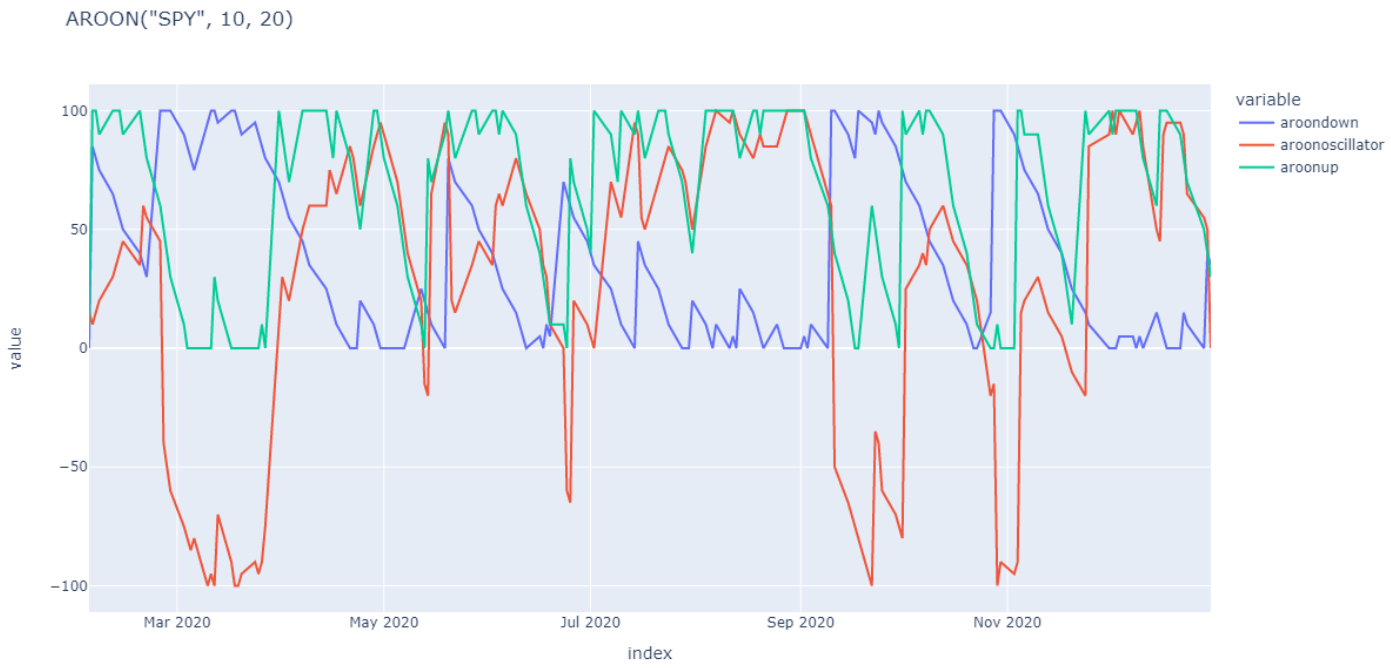
Return

`AroonOscillator` - The new `AroonOscillator` indicator object.

Definition at line 65 of file Indicators/AroonOscillator.cs.

Visualization

The following image shows plot values of selected properties of `AroonOscillator` using the plotly library.



Supported Indicators

Augen Price Spike

Introduction

The Augen Price Spike indicator is an indicator that measures price changes in terms of standard deviations. In the book, The Volatility Edge in Options Trading, Jeff Augen describes a method for tracking absolute price changes in terms of recent volatility, using the standard deviation. length = x closes = closeArray closes1 = closeArray shifted right by 1 closes2 = closeArray shifted right by 2 closeLog = np.log(np.divide(closes1, closes2)) SDev = np.std(closeLog) m = SDev * closes1[-1] spike = (closes[-1]-closes1[-1])/m return spike Augen Price Spike from TradingView https://www.tradingview.com/script/fC7Pn2X2-Price-Spike-Jeff-Augen/

To view the implementation of this indicator, see the LEAN GitHub repository .

Using APS Indicator

To create an automatic indicators for `AugenPriceSpike` , call the `APS` helper method from the `QCAlgorithm` class. The `APS` method creates a `AugenPriceSpike` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class AugenPriceSpikeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.aps = self.APS(self.symbol, 3)

    def OnData(self, slice: Slice) -> None:
        if self.aps.IsReady:
            # The current value of self.aps is represented by self.aps.Current.Value
            self.Plot("AugenPriceSpike", "aps", self.aps.Current.Value)
```

The following reference table describes the `APS` method:

INDICATORS

APS() 1/1

```
AugenPriceSpike QuantConnect.Algorithm.QCAlgorithm.APS (
Symbol symbol,
*Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an AugenPriceSpike indicator for the symbol. The indicator will be automatically updated on the given

resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose APS we want.
`*Int32`	period	*(Optional)* The period of the APS.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AugenPriceSpike` - The AugenPriceSpike indicator for the given parameters.

Definition at line 277 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AugenPriceSpike` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class AugenPriceSpikeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.aps = AugenPriceSpike(3)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.aps.Update(bar.EndTime, bar.Close)

        if self.aps.IsReady:
            # The current value of self.aps is represented by self.aps.Current.Value
            self.Plot("AugenPriceSpike", "aps", self.aps.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class AugenPriceSpikeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.aps = AugenPriceSpike(3)
        self.RegisterIndicator(self.symbol, self.aps, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.aps.IsReady:
            # The current value of self.aps is represented by self.aps.Current.Value
            self.Plot("AugenPriceSpike", "aps", self.aps.Current.Value)
```

The following reference table describes the `AugenPriceSpike` constructor:

INDICATORS

AugenPriceSpike() 1/2

```
AugenPriceSpike QuantConnect.Indicators.AugenPriceSpike (
*int period
)
```

Initializes a new instance of the AugenPriceSpike class using the specified period.

Show Details

Parameters		
*int	period	*(Optional) (Optional)* The period over which to perform to computation. Default: 3.

Return

`AugenPriceSpike` - The new `AugenPriceSpike` indicator object.

Definition at line 49 of file Indicators/AugenPriceSpike.cs.

INDICATORS

AugenPriceSpike() 2/2

```
AugenPriceSpike QuantConnect.Indicators.AugenPriceSpike (
string name,
int period
)
```

Creates a new AugenPriceSpike indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of this indicator.

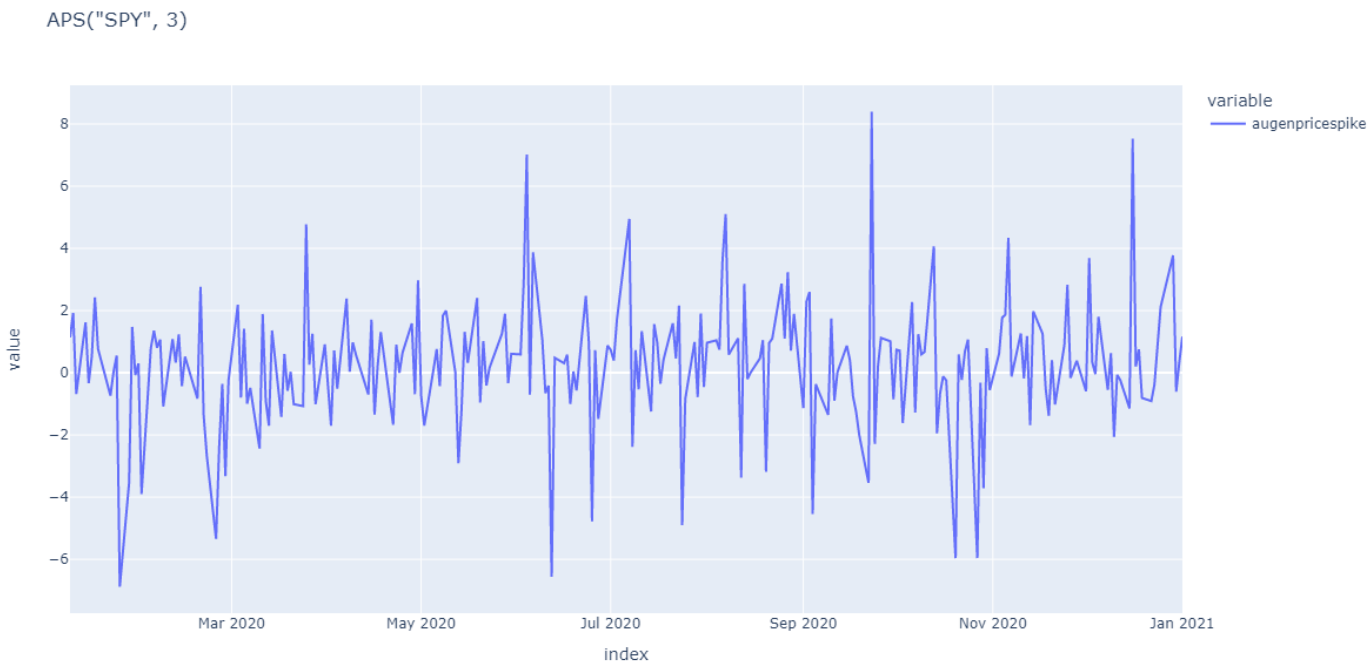
Return

`AugenPriceSpike` - The new `AugenPriceSpike` indicator object.

Definition at line 58 of file Indicators/AugenPriceSpike.cs.

Visualization

The following image shows plot values of selected properties of `AugenPriceSpike` using the plotly library.



Supported Indicators

Auto Regressive Integrated Moving Average

Introduction

An Autoregressive Intergrated Moving Average (ARIMA) is a time series model which can be used to describe a set of data. In particular,with X_t representing the series, the model assumes the data are of form (after differencing times): $X_t = c + \varepsilon_t + \Sigma_i \varphi_i X_{t-i} + \Sigma_i \theta_i \varepsilon_{t-i}$ where the first sum has an upper limit of and the second .

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ARIMA Indicator

To create an automatic indicators for `AutoRegressiveIntegratedMovingAverage` , call the `ARIMA` helper method from the `QCAlgorithm` class. The `ARIMA` method creates a `AutoRegressiveIntegratedMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AutoRegressiveIntegratedMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.arima = self.ARIMA(self.symbol, 1, 1, 1, 20)

    def OnData(self, slice: Slice) -> None:
        if self.arima.IsReady:
            # The current value of self.arima is represented by self.arima.Current.Value
            self.Plot("AutoRegressiveIntegratedMovingAverage", "arima", self.arima.Current.Value)
            # Plot all attributes of self.arima
            self.Plot("AutoRegressiveIntegratedMovingAverage", "arresidualerror",
self.arima.ArResidualError.Current.Value)
            self.Plot("AutoRegressiveIntegratedMovingAverage", "maresidualerror",
self.arima.MaResidualError.Current.Value)
```

The following reference table describes the `ARIMA` method:

INDICATORS

ARIMA() 1/1

```
AutoRegressiveIntegratedMovingAverage QuantConnect.Algorithm.QCAlgorithm.ARIMA (
Symbol symbol,
Int32 arOrder,
Int32 diffOrder,
Int32 maOrder,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new ARIMA indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ARIMA indicator we want.
`Int32`	arOrder	AR order (p) -- defines the number of past values to consider in the AR component of the model.
`Int32`	diffOrder	Difference order (d) -- defines how many times to difference the model before fitting parameters.
`Int32`	maOrder	MA order (q) -- defines the number of past values to consider in the MA component of the model.
`Int32`	period	Size of the rolling series to fit onto.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AutoRegressiveIntegratedMovingAverage` - The ARIMA indicator for the requested symbol over the specified period.

Definition at line 104 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AutoRegressiveIntegratedMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```
class AutoRegressiveIntegratedMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.arima = AutoRegressiveIntegratedMovingAverage(1, 1, 1, 20, True)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.arima.Update(bar.EndTime, bar.Close)

        if self.arima.IsReady:
            # The current value of self.arima is represented by self.arima.Current.Value
            self.Plot("AutoRegressiveIntegratedMovingAverage", "arima", self.arima.Current.Value)
            # Plot all attributes of self.arima
            self.Plot("AutoRegressiveIntegratedMovingAverage", "arresidualerror",
self.arima.ArResidualError.Current.Value)
            self.Plot("AutoRegressiveIntegratedMovingAverage", "maresidualerror",
self.arima.MaResidualError.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class AutoRegressiveIntegratedMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.arima = AutoRegressiveIntegratedMovingAverage(1, 1, 1, 20, True)
        self.RegisterIndicator(self.symbol, self.arima, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.arima.IsReady:
            # The current value of self.arima is represented by self.arima.Current.Value
            self.Plot("AutoRegressiveIntegratedMovingAverage", "arima", self.arima.Current.Value)
            # Plot all attributes of self.arima
            self.Plot("AutoRegressiveIntegratedMovingAverage", "arresidualerror",
self.arima.ArResidualError.Current.Value)
            self.Plot("AutoRegressiveIntegratedMovingAverage", "maresidualerror",
self.arima.MaResidualError.Current.Value)
```

The following reference table describes the `AutoRegressiveIntegratedMovingAverage` constructor:

INDICATORS

AutoRegressiveIntegratedMovingAverage() 1/2

```
AutoRegressiveIntegratedMovingAverage QuantConnect.Indicators.AutoRegressiveIntegratedMovingAverage (
string name,
int arOrder,
int diffOrder,
int maOrder,
int period,
*bool intercept
)
```

This particular constructor fits the model by means of `TwoStepFit` for a specified name.

Show Details

Parameters		
`string`	name	The name of the indicator.
`int`	arOrder	AR order (p) -- defines the number of past values to consider in the AR component of the model.
`int`	diffOrder	Difference order (d) -- defines how many times to difference the model before fitting parameters.
`int`	maOrder	MA order -- defines the number of past values to consider in the MA component of the model.
`int`	period	Size of the rolling series to fit onto.
`*bool`	intercept	*(Optional) (Optional)* Whether ot not to include the intercept term. Default: true.

Return

`AutoRegressiveIntegratedMovingAverage` - The new `AutoRegressiveIntegratedMovingAverage` indicator object.

Definition at line 105 of file Indicators/AutoRegressiveIntegratedMovingAverage.cs.

INDICATORS

AutoRegressiveIntegratedMovingAverage() 2/2

```
AutoRegressiveIntegratedMovingAverage QuantConnect.Indicators.AutoRegressiveIntegratedMovingAverage (
int arOrder,
int diffOrder,
int maOrder,
int period,
bool intercept
)
```

This particular constructor fits the model by means of `TwoStepFit` using ordinary least squares.

Show Details

Parameters		
`int`	arOrder	AR order (p) -- defines the number of past values to consider in the AR component of the model.
`int`	diffOrder	Difference order (d) -- defines how many times to difference the model before fitting parameters.
`int`	maOrder	MA order (q) -- defines the number of past values to consider in the MA component of the model.
`int`	period	Size of the rolling series to fit onto.
`bool`	intercept	Whether to include an intercept term (c).

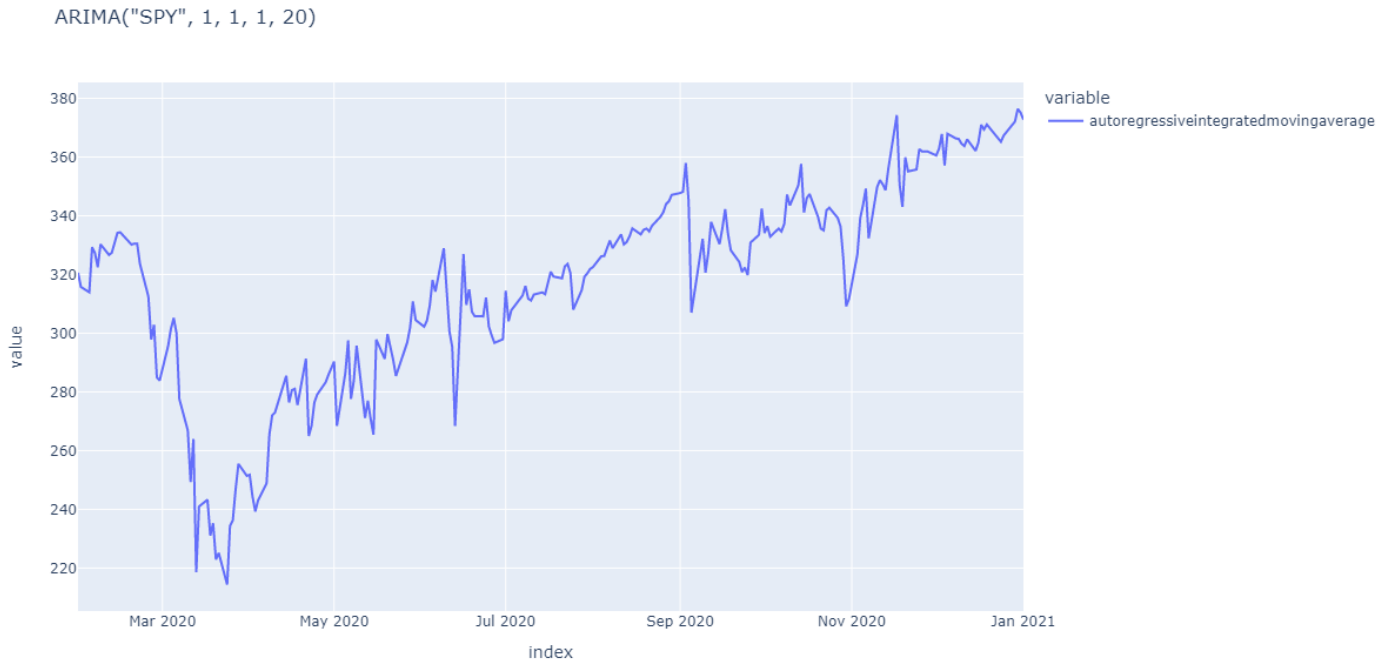
Return

`AutoRegressiveIntegratedMovingAverage` - The new `AutoRegressiveIntegratedMovingAverage` indicator object.

Definition at line 152 of file Indicators/AutoRegressiveIntegratedMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `AutoRegressiveIntegratedMovingAverage` using the plotly library.



Supported Indicators

Average Directional Index

Introduction

This indicator computes Average Directional Index which measures trend strength without regard to trend direction. Firstly, it calculates the Directional Movement and the True Range value, and then the values are accumulated and smoothed using a custom smoothing method proposed by Wilder. For an n period smoothing, 1/n of each period's value is added to the total period. From these accumulated values we are therefore able to derived the 'Positive Directional Index' (+DI) and 'Negative Directional Index' (-DI) which is used to calculate the Average Directional Index. Computation source: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:average_directional_index_adx

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADX Indicator

To create an automatic indicators for `AverageDirectionalIndex` , call the `ADX` helper method from the `QCAlgorithm` class. The `ADX` method creates a `AverageDirectionalIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AverageDirectionalIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adx = self.ADX(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.adx.IsReady:
            # The current value of self.adx is represented by self.adx.Current.Value
            self.Plot("AverageDirectionalIndex", "adx", self.adx.Current.Value)
            # Plot all attributes of self.adx
            self.Plot("AverageDirectionalIndex", "positivedirectionalindex",
self.adx.PositiveDirectionalIndex.Current.Value)
            self.Plot("AverageDirectionalIndex", "negativedirectionalindex",
self.adx.NegativeDirectionalIndex.Current.Value)
```

The following reference table describes the `ADX` method:

INDICATORS

ADX() 1/1

```
AverageDirectionalIndex QuantConnect.Algorithm.QCAlgorithm.ADX (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Average Directional Index indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Average Directional Index we seek.
`Int32`	period	The period over which to compute the Average Directional Index.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AverageDirectionalIndex` - The Average Directional Index indicator for the requested symbol.

Definition at line 124 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AverageDirectionalIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```
class AverageDirectionalIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adx = AverageDirectionalIndex(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.adx.Update(bar)

        if self.adx.IsReady:
            # The current value of self.adx is represented by self.adx.Current.Value
            self.Plot("AverageDirectionalIndex", "adx", self.adx.Current.Value)
            # Plot all attributes of self.adx
            self.Plot("AverageDirectionalIndex", "positivedirectionalindex",
self.adx.PositiveDirectionalIndex.Current.Value)
            self.Plot("AverageDirectionalIndex", "negativedirectionalindex",
self.adx.NegativeDirectionalIndex.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class AverageDirectionalIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adx = AverageDirectionalIndex(20)
        self.RegisterIndicator(self.symbol, self.adx, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.adx.IsReady:
            # The current value of self.adx is represented by self.adx.Current.Value
            self.Plot("AverageDirectionalIndex", "adx", self.adx.Current.Value)
            # Plot all attributes of self.adx
            self.Plot("AverageDirectionalIndex", "positivedirectionalindex",
self.adx.PositiveDirectionalIndex.Current.Value)
            self.Plot("AverageDirectionalIndex", "negativedirectionalindex",
self.adx.NegativeDirectionalIndex.Current.Value)
```

The following reference table describes the `AverageDirectionalIndex` constructor:

INDICATORS

AverageDirectionalIndex() 1/2

```
AverageDirectionalIndex QuantConnect.Indicators.AverageDirectionalIndex (
int period
)
```

Initializes a new instance of the `AverageDirectionalInde` class.

Show Details

Parameters		
int	period	The period.

Return

`AverageDirectionalIndex` - The new `AverageDirectionalIndex` indicator object.

Definition at line 72 of file Indicators/AverageDirectionalIndex.cs.

INDICATORS

AverageDirectionalIndex() 2/2

```
AverageDirectionalIndex QuantConnect.Indicators.AverageDirectionalIndex (
string name,
int period
)
```

Initializes a new instance of the `AverageDirectionalInde` class.

Show Details

Parameters		
`string`	name	The name.
`int`	period	The period.

Return

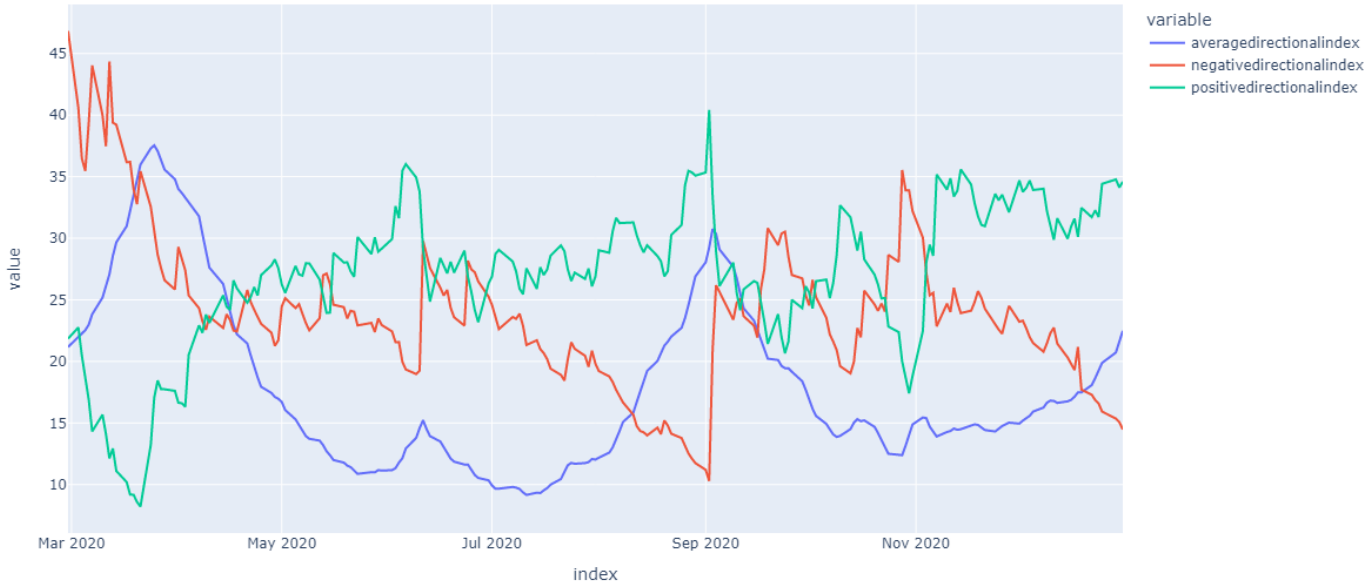
`AverageDirectionalIndex` - The new `AverageDirectionalIndex` indicator object.

Definition at line 82 of file Indicators/AverageDirectionalIndex.cs.

Visualization

The following image shows plot values of selected properties of `AverageDirectionalIndex` using the plotly library.

ADX("SPY", 20)


variable
averagedirectionalindex
negativedirectionalindex
positivedirectionalindex
value
45
40
35
30
25
20
15
10
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
index

Supported Indicators

Average Directional Movement Index Rating

Introduction

This indicator computes the Average Directional Movement Index Rating (ADXR). The Average Directional Movement Index Rating is calculated with the following formula: ADXR[i] = (ADX[i] + ADX[i - period + 1]) / 2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADXR Indicator

To create an automatic indicators for `AverageDirectionalMovementIndexRating` , call the `ADXR` helper method from the `QCAlgorithm` class. The `ADXR` method creates a `AverageDirectionalMovementIndexRating` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class AverageDirectionalMovementIndexRatingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adxr = self.ADXR(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.adxr.IsReady:
            # The current value of self.adxr is represented by self.adxr.Current.Value
            self.Plot("AverageDirectionalMovementIndexRating", "adxr", self.adxr.Current.Value)
            # Plot all attributes of self.adxr
            self.Plot("AverageDirectionalMovementIndexRating", "adx", self.adxr.ADX.Current.Value)
```

The following reference table describes the `ADXR` method:

INDICATORS

ADXR() 1/1

```
AverageDirectionalMovementIndexRating QuantConnect.Algorithm.QCAlgorithm.ADXR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AverageDirectionalMovementIndexRating indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ADXR we want.
`Int32`	period	The period over which to compute the ADXR.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AverageDirectionalMovementIndexRating` - The AverageDirectionalMovementIndexRating indicator for the requested symbol over the specified period.

Definition at line 161 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AverageDirectionalMovementIndexRating` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class AverageDirectionalMovementIndexRatingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adxr = AverageDirectionalMovementIndexRating(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.adxr.Update(bar)

        if self.adxr.IsReady:
            # The current value of self.adxr is represented by self.adxr.Current.Value
            self.Plot("AverageDirectionalMovementIndexRating", "adxr", self.adxr.Current.Value)
            # Plot all attributes of self.adxr
            self.Plot("AverageDirectionalMovementIndexRating", "adx", self.adxr.ADX.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class AverageDirectionalMovementIndexRatingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.adxr = AverageDirectionalMovementIndexRating(20)
        self.RegisterIndicator(self.symbol, self.adxr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.adxr.IsReady:
            # The current value of self.adxr is represented by self.adxr.Current.Value
            self.Plot("AverageDirectionalMovementIndexRating", "adxr", self.adxr.Current.Value)
            # Plot all attributes of self.adxr
            self.Plot("AverageDirectionalMovementIndexRating", "adx", self.adxr.ADX.Current.Value)
```

The following reference table describes the `AverageDirectionalMovementIndexRating` constructor:

INDICATORS

AverageDirectionalMovementIndexRating() 1/2

```
AverageDirectionalMovementIndexRating QuantConnect.Indicators.AverageDirectionalMovementIndexRating (
string name,
int period
)
```

Initializes a new instance of the `AverageDirectionalMovementIndexRatin` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the ADXR.

Return

`AverageDirectionalMovementIndexRating` - The new `AverageDirectionalMovementIndexRating` indicator object.

Definition at line 35 of file Indicators/AverageDirectionalMovementIndexRating.cs.

INDICATORS

AverageDirectionalMovementIndexRating() 2/2

```
AverageDirectionalMovementIndexRating QuantConnect.Indicators.AverageDirectionalMovementIndexRating (
int period
)
```

Initializes a new instance of the `AverageDirectionalMovementIndexRatin` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the ADXR.

Return

`AverageDirectionalMovementIndexRating` - The new `AverageDirectionalMovementIndexRating` indicator object.

Definition at line 47 of file Indicators/AverageDirectionalMovementIndexRating.cs.

Visualization

The following image shows plot values of selected properties of `AverageDirectionalMovementIndexRating` using the plotly library.

Supported Indicators

Average True Range

Introduction

The AverageTrueRange indicator is a measure of volatility introduced by Welles Wilder in his book: New Concepts in Technical Trading Systems. This indicator computes the TrueRange and then smoothes the TrueRange over a given period. TrueRange is defined as the maximum of the following: High - Low ABS(High - PreviousClose) ABS(Low - PreviousClose)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ATR Indicator

To create an automatic indicators for `AverageTrueRange` , call the `ATR` helper method from the `QCAlgorithm` class. The `ATR` method creates a `AverageTrueRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class AverageTrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.atr = self.ATR(self.symbol, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.atr.IsReady:
            # The current value of self.atr is represented by self.atr.Current.Value
            self.Plot("AverageTrueRange", "atr", self.atr.Current.Value)
            # Plot all attributes of self.atr
            self.Plot("AverageTrueRange", "truerange", self.atr.TrueRange.Current.Value)
```

The following reference table describes the `ATR` method:

INDICATORS

ATR() 1/1

```
AverageTrueRange QuantConnect.Algorithm.QCAlgorithm.ATR (
Symbol symbol,
Int32 period,
*MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AverageTrueRange indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ATR we want.
`Int32`	period	The smoothing period used to smooth the computed TrueRange values.
`*MovingAverageType`	type	*(Optional)* The type of smoothing to use.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AverageTrueRange` - A new AverageTrueRange indicator with the specified smoothing type and period.

Definition at line 258 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `AverageTrueRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class AverageTrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.atr = AverageTrueRange(20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.atr.Update(bar)

        if self.atr.IsReady:
            # The current value of self.atr is represented by self.atr.Current.Value
            self.Plot("AverageTrueRange", "atr", self.atr.Current.Value)
            # Plot all attributes of self.atr
            self.Plot("AverageTrueRange", "truerange", self.atr.TrueRange.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AverageTrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.atr = AverageTrueRange(20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.atr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.atr.IsReady:
            # The current value of self.atr is represented by self.atr.Current.Value
            self.Plot("AverageTrueRange", "atr", self.atr.Current.Value)
            # Plot all attributes of self.atr
            self.Plot("AverageTrueRange", "truerange", self.atr.TrueRange.Current.Value)
```

The following reference table describes the `AverageTrueRange` constructor:

INDICATORS

AverageTrueRange() 1/2

```
AverageTrueRange QuantConnect.Indicators.AverageTrueRange (
string name,
int period,
*MovingAverageType movingAverageType
)
```

Creates a new AverageTrueRange indicator using the specified period and moving average type.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The smoothing period used to smooth the true range values.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of smoothing used to smooth the true range values. Default: MovingAverageType.Wilders.

Return

AverageTrueRange - The new AverageTrueRange indicator object.

Definition at line 61 of file Indicators/AverageTrueRange.cs.

INDICATORS

AverageTrueRange() 2/2

```
AverageTrueRange QuantConnect.Indicators.AverageTrueRange (
int period,
*MovingAverageType movingAverageType
)
```

Creates a new AverageTrueRange indicator using the specified period and moving average type.

Show Details

Parameters		
int	period	The smoothing period used to smooth the true range values.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of smoothing used to smooth the true range values. Default: MovingAverageType.Wilders.

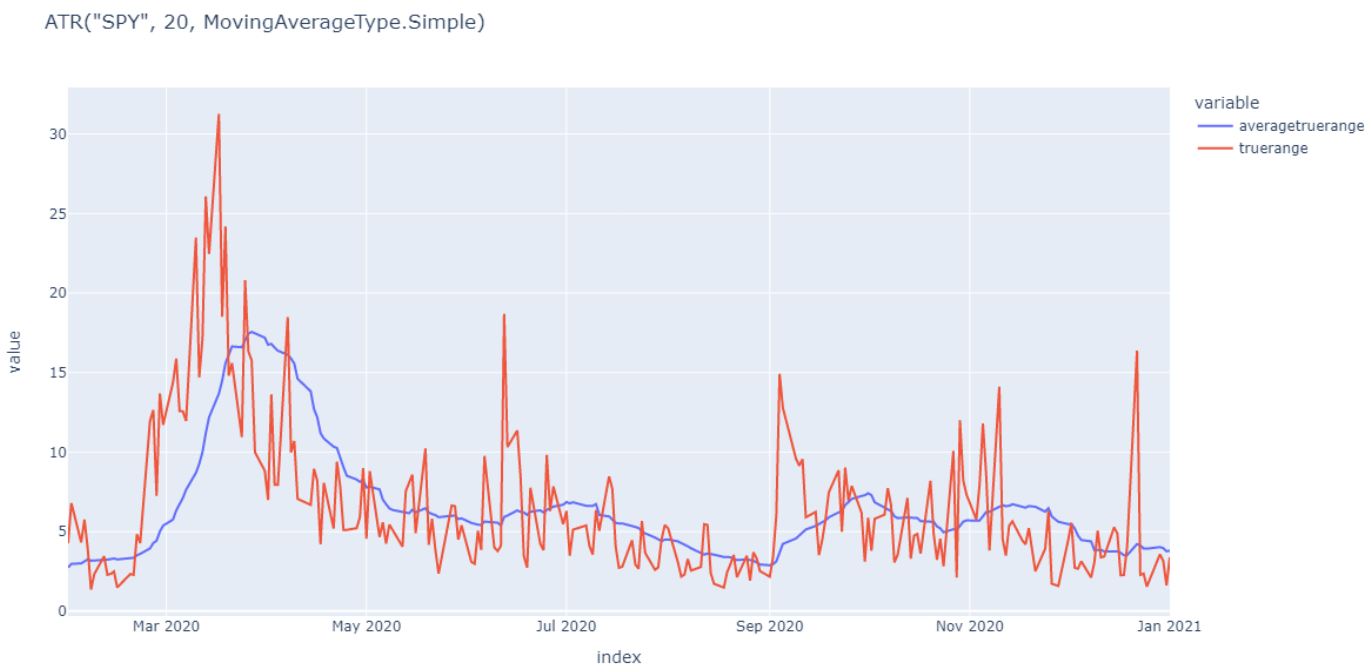
Return

AverageTrueRange - The new AverageTrueRange indicator object.

Definition at line 84 of file Indicators/AverageTrueRange.cs.

Visualization

The following image shows plot values of selected properties of `AverageTrueRange` using the plotly library.



Supported Indicators

Awesome Oscillator

Introduction

The Awesome Oscillator Indicator tracks the price midpoint-movement of a security. Specifically, AO = MAfast[(H+L)/2] - MAslow[(H+L)/2] where MAfast and MAslow denote simple moving averages wherein fast has a shorter period. https://www.barchart.com/education/technical-indicators/awesome_oscillator

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AO Indicator

To create an automatic indicators for `AwesomeOscillator` , call the `AO` helper method from the `QCAlgorithm` class. The `AO` method creates a `AwesomeOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class AwesomeOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ao = self.AO(self.symbol, 10, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.ao.IsReady:
            # The current value of self.ao is represented by self.ao.Current.Value
            self.Plot("AwesomeOscillator", "ao", self.ao.Current.Value)
            # Plot all attributes of self.ao
            self.Plot("AwesomeOscillator", "slowao", self.ao.SlowAo.Current.Value)
            self.Plot("AwesomeOscillator", "fastao", self.ao.FastAo.Current.Value)
```

The following reference table describes the `AO` method:

INDICATORS

AO() 1/1

```
AwesomeOscillator QuantConnect.Algorithm.QCAlgorithm.AO (
Symbol symbol,
Int32 slowPeriod,
Int32 fastPeriod,
MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Awesome Oscillator from the specified periods.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Awesome Oscillator we seek.
`Int32`	slowPeriod	The period of the slow moving average associated with the AO.
`Int32`	fastPeriod	The period of the fast moving average associated with the AO.
`MovingAverageType`	type	The type of moving average used when computing the fast and slow term. Defaults to simple moving average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AwesomeOscillator` - The new `AwesomeOscillator` object.

Definition at line 143 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `AwesomeOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class AwesomeOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ao = AwesomeOscillator(10, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ao.Update(bar)

        if self.ao.IsReady:
            # The current value of self.ao is represented by self.ao.Current.Value
            self.Plot("AwesomeOscillator", "ao", self.ao.Current.Value)
            # Plot all attributes of self.ao
            self.Plot("AwesomeOscillator", "slowao", self.ao.SlowAo.Current.Value)
            self.Plot("AwesomeOscillator", "fastao", self.ao.FastAo.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class AwesomeOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ao = AwesomeOscillator(10, 20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.ao, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ao.IsReady:
            # The current value of self.ao is represented by self.ao.Current.Value
            self.Plot("AwesomeOscillator", "ao", self.ao.Current.Value)
            # Plot all attributes of self.ao
            self.Plot("AwesomeOscillator", "slowao", self.ao.SlowAo.Current.Value)
            self.Plot("AwesomeOscillator", "fastao", self.ao.FastAo.Current.Value)
```

The following reference table describes the `AwesomeOscillator` constructor:

INDICATORS

AwesomeOscillator() 1/2

```
AwesomeOscillator QuantConnect.Indicators.AwesomeOscillator (
int fastPeriod,
int slowPeriod
)
```

Creates a new Awesome Oscillator from the specified periods.

Show Details

Parameters		
int	fastPeriod	The period of the fast moving average associated with the AO.
int	slowPeriod	The period of the slow moving average associated with the AO.

Return

`AwesomeOscillator` - The new `AwesomeOscillator` indicator object.

Definition at line 57 of file Indicators/AwesomeOscillator.cs.

INDICATORS

AwesomeOscillator() 2/2

```
AwesomeOscillator QuantConnect.Indicators.AwesomeOscillator (
string name,
int fastPeriod,
int slowPeriod
)
```

Creates a new Awesome Oscillator from the specified periods.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	fastPeriod	The period of the fast moving average associated with the AO.
`int`	slowPeriod	The period of the slow moving average associated with the AO.

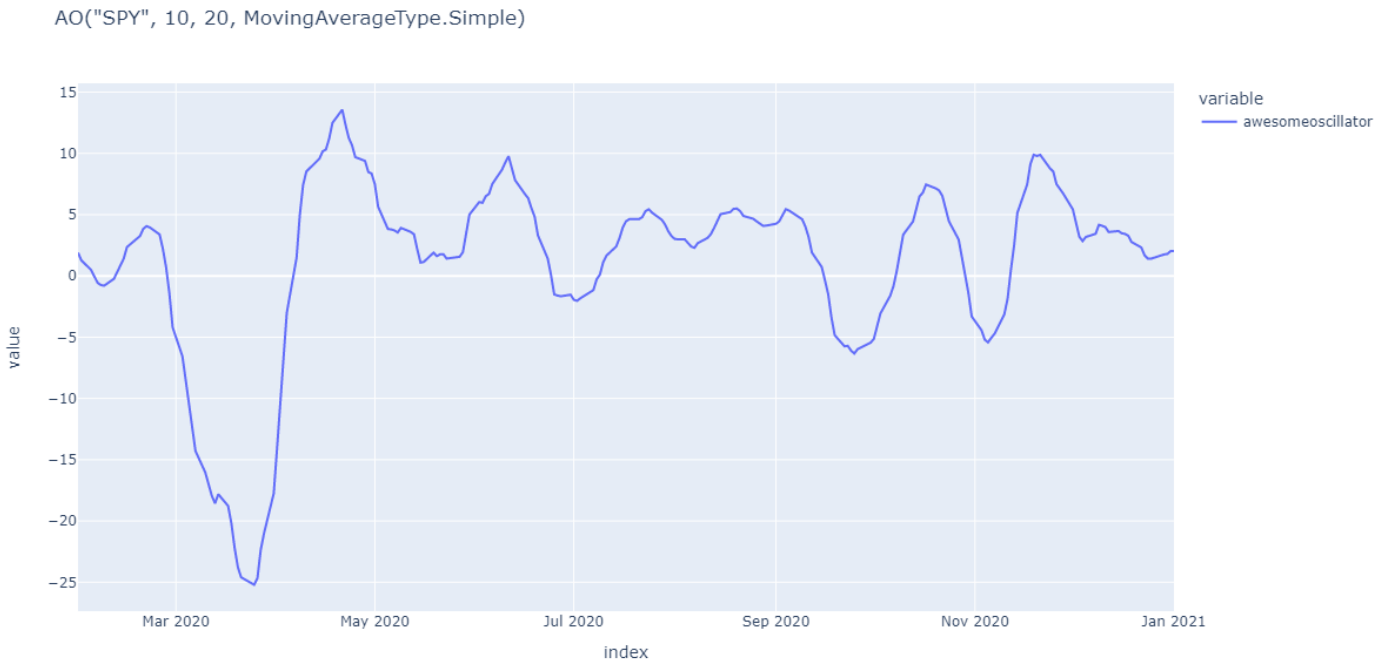
Return

`AwesomeOscillator` - The new `AwesomeOscillator` indicator object.

Definition at line 69 of file Indicators/AwesomeOscillator.cs.

Visualization

The following image shows plot values of selected properties of `AwesomeOscillator` using the plotly library.


AO("SPY", 10, 20, MovingAverageType.Simple)
variable
awesomeoscillator
value
15
10
5
0
−5
−10
−15
−20
−25
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Balance Of Power

Introduction

This indicator computes the Balance Of Power (BOP). The Balance Of Power is calculated with the following formula: BOP = (Close - Open) / (High - Low)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using BOP Indicator

To create an automatic indicators for `BalanceOfPower` , call the `BOP` helper method from the `QCAlgorithm` class. The `BOP` method creates a `BalanceOfPower` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class BalanceOfPowerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bop = self.BOP(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.bop.IsReady:
            # The current value of self.bop is represented by self.bop.Current.Value
            self.Plot("BalanceOfPower", "bop", self.bop.Current.Value)
```

The following reference table describes the `BOP` method:

INDICATORS

BOP() 1/1

```
BalanceOfPower QuantConnect.Algorithm.QCAlgorithm.BOP (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Balance Of Power indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Balance Of Power we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`BalanceOfPower` - The Balance Of Power indicator for the requested symbol.

Definition at line 337 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `BalanceOfPower` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class BalanceOfPowerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bop = BalanceOfPower()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.bop.Update(bar)

        if self.bop.IsReady:
            # The current value of self.bop is represented by self.bop.Current.Value
            self.Plot("BalanceOfPower", "bop", self.bop.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class BalanceOfPowerAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bop = BalanceOfPower()
        self.RegisterIndicator(self.symbol, self.bop, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.bop.IsReady:
            # The current value of self.bop is represented by self.bop.Current.Value
            self.Plot("BalanceOfPower", "bop", self.bop.Current.Value)
```

The following reference table describes the `BalanceOfPower` constructor:

INDICATORS

BalanceOfPower() 1/2

```
BalanceOfPower QuantConnect.Indicators.BalanceOfPower (

)
```

Initializes a new instance of the `BalanceOfPowe` class using the specified name.

Show Details

This method requires no argument input.

Return

`BalanceOfPower` - The new `BalanceOfPower` indicator object.

Definition at line 30 of file Indicators/BalanceOfPower.cs.

INDICATORS

BalanceOfPower() 2/2

```
BalanceOfPower QuantConnect.Indicators.BalanceOfPower (
string name
)
```

Initializes a new instance of the `BalanceOfPowe` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

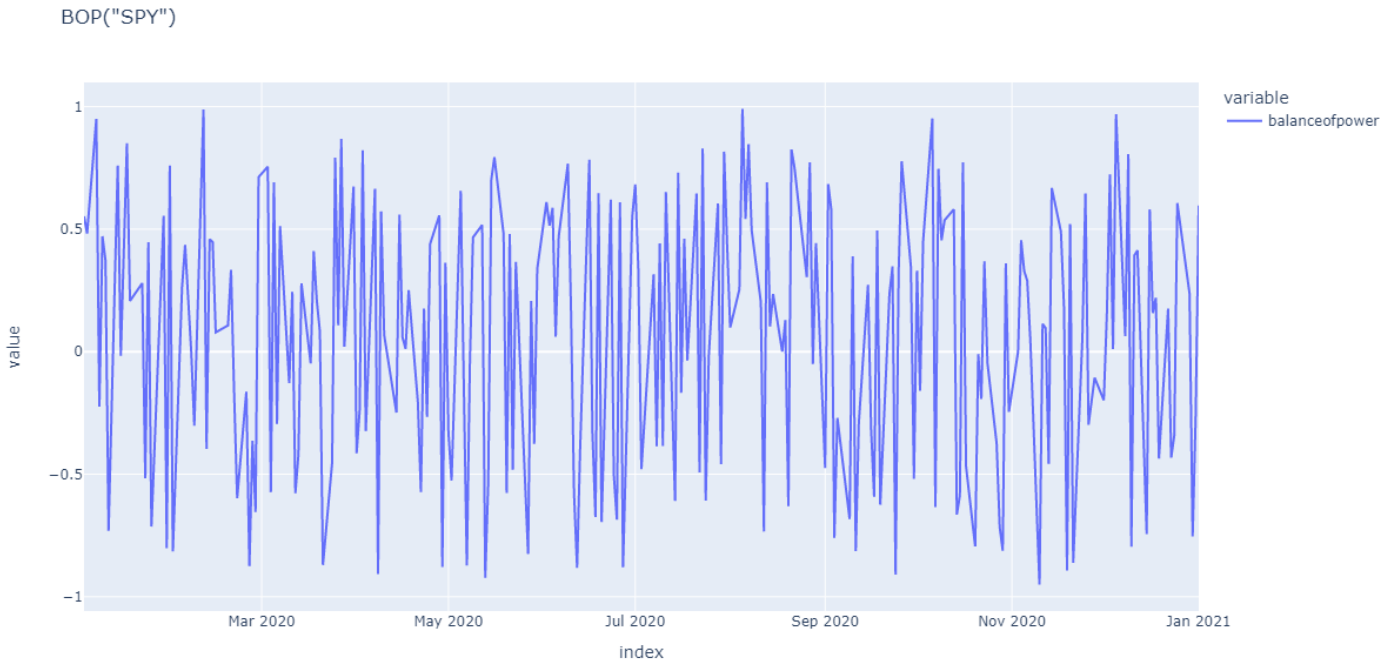
Return

`BalanceOfPower` - The new `BalanceOfPower` indicator object.

Definition at line 39 of file Indicators/BalanceOfPower.cs.

Visualization

The following image shows plot values of selected properties of `BalanceOfPower` using the plotly library.



Supported Indicators

Beta

Introduction

In technical analysis Beta indicator is used to measure volatility or risk of a target (ETF) relative to the overall risk (volatility) of the reference (market indexes). The Beta indicators compares target's price movement to the movements of the indexes over the same period of time. It is common practice to use the SPX index as a benchmark of the overall reference market when it comes to Beta calculations.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using B Indicator

To create an automatic indicators for `Beta` , call the `B` helper method from the `QCAlgorithm` class. The `B` method creates a `Beta` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class BetaAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.b = self.B(Symbol.Create("QQQ", SecurityType.Equity, Market.USA), self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.b.IsReady:
            # The current value of self.b is represented by self.b.Current.Value
            self.Plot("Beta", "b", self.b.Current.Value)
```

The following reference table describes the `B` method:

INDICATORS

B() 1/1

```
Beta QuantConnect.Algorithm.QCAlgorithm.B (
Symbol target,
Symbol reference,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a Beta indicator for the given target symbol in relation with the reference used. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	target	The target symbol whose Beta value we want.
`Symbol`	reference	The reference symbol to compare with the target symbol.
`Int32`	period	The period of the Beta indicator.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`Beta` - The Beta indicator for the given parameters.

Definition at line 318 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Beta` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class BetaAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.b = Beta("", 20, Symbol.Create("QQQ", SecurityType.Equity, Market.USA), Symbol.Create("SPY",
SecurityType.Equity, Market.USA))

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.b.Update(bar)

        if self.b.IsReady:
            # The current value of self.b is represented by self.b.Current.Value
            self.Plot("Beta", "b", self.b.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class BetaAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.b = Beta("", 20, Symbol.Create("QQQ", SecurityType.Equity, Market.USA), Symbol.Create("SPY",
SecurityType.Equity, Market.USA))
        self.RegisterIndicator(self.symbol, self.b, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.b.IsReady:
            # The current value of self.b is represented by self.b.Current.Value
            self.Plot("Beta", "b", self.b.Current.Value)
```

The following reference table describes the `Beta` constructor:

INDICATORS

Beta() 1/1

```
Beta QuantConnect.Indicators.Beta (
string name,
int period,
Symbol targetSymbol,
Symbol referenceSymbol
)
```

reference values.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of this indicator.
`Symbol`	targetSymbol	The target symbol of this indicator.
`Symbol`	referenceSymbol	The reference symbol of this indicator.

Return

`Beta` - The new `Beta` indicator object.

Definition at line 85 of file Indicators/Beta.cs.

Visualization

The following image shows plot values of selected properties of `Beta` using the plotly library.

Supported Indicators

Bollinger Bands

Introduction

This indicator creates a moving average (middle band) with an upper band and lower band fixed at k standard deviations above and below the moving average.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using BB Indicator

To create an automatic indicators for `BollingerBands` , call the `BB` helper method from the `QCAlgorithm` class. The `BB` method creates a `BollingerBands` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class BollingerBandsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bb = self.BB(self.symbol, 30, 2)

    def OnData(self, slice: Slice) -> None:
        if self.bb.IsReady:
            # The current value of self.bb is represented by self.bb.Current.Value
            self.Plot("BollingerBands", "bb", self.bb.Current.Value)
            # Plot all attributes of self.bb
            self.Plot("BollingerBands", "standarddeviation", self.bb.StandardDeviation.Current.Value)
            self.Plot("BollingerBands", "middleband", self.bb.MiddleBand.Current.Value)
            self.Plot("BollingerBands", "upperband", self.bb.UpperBand.Current.Value)
            self.Plot("BollingerBands", "lowerband", self.bb.LowerBand.Current.Value)
            self.Plot("BollingerBands", "bandwidth", self.bb.BandWidth.Current.Value)
            self.Plot("BollingerBands", "percentb", self.bb.PercentB.Current.Value)
            self.Plot("BollingerBands", "price", self.bb.Price.Current.Value)
```

The following reference table describes the `BB` method:

INDICATORS

BB() 1/1

```
BollingerBands QuantConnect.Algorithm.QCAlgorithm.BB (
Symbol symbol,
Int32 period,
Decimal k,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new BollingerBands indicator which will compute the MiddleBand, UpperBand, LowerBand, and StandardDeviation.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose BollingerBands we seek.
`Int32`	period	The period of the standard deviation and moving average (middle band).
`Decimal`	k	The number of standard deviations specifying the distance between the middle band and upper or lower bands.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to be used.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`BollingerBands` - A BollingerBands configured with the specified period.

Definition at line 297 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `BollingerBands` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class BollingerBandsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bb = BollingerBands(30, 2)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.bb.Update(bar.EndTime, bar.Close)

        if self.bb.IsReady:
            # The current value of self.bb is represented by self.bb.Current.Value
            self.Plot("BollingerBands", "bb", self.bb.Current.Value)
            # Plot all attributes of self.bb
            self.Plot("BollingerBands", "standarddeviation", self.bb.StandardDeviation.Current.Value)
            self.Plot("BollingerBands", "middleband", self.bb.MiddleBand.Current.Value)
            self.Plot("BollingerBands", "upperband", self.bb.UpperBand.Current.Value)
            self.Plot("BollingerBands", "lowerband", self.bb.LowerBand.Current.Value)
            self.Plot("BollingerBands", "bandwidth", self.bb.BandWidth.Current.Value)
            self.Plot("BollingerBands", "percentb", self.bb.PercentB.Current.Value)
            self.Plot("BollingerBands", "price", self.bb.Price.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class BollingerBandsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bb = BollingerBands(30, 2)
        self.RegisterIndicator(self.symbol, self.bb, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.bb.IsReady:
            # The current value of self.bb is represented by self.bb.Current.Value
            self.Plot("BollingerBands", "bb", self.bb.Current.Value)
            # Plot all attributes of self.bb
            self.Plot("BollingerBands", "standarddeviation", self.bb.StandardDeviation.Current.Value)
            self.Plot("BollingerBands", "middleband", self.bb.MiddleBand.Current.Value)
            self.Plot("BollingerBands", "upperband", self.bb.UpperBand.Current.Value)
            self.Plot("BollingerBands", "lowerband", self.bb.LowerBand.Current.Value)
            self.Plot("BollingerBands", "bandwidth", self.bb.BandWidth.Current.Value)
            self.Plot("BollingerBands", "percentb", self.bb.PercentB.Current.Value)
            self.Plot("BollingerBands", "price", self.bb.Price.Current.Value)
```

The following reference table describes the `BollingerBands` constructor:

INDICATORS

BollingerBands() 1/2

```
BollingerBands QuantConnect.Indicators.BollingerBands (
int period,
decimal k,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the BollingerBands class.

Show Details

Parameters		
int	period	The period of the standard deviation and moving average (middle band).
decimal	k	The number of standard deviations specifying the distance between the middle band and upper or lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used. Default: MovingAverageType.Simple.

Return

`BollingerBands` - The new `BollingerBands` indicator object.

Definition at line 72 of file Indicators/BollingerBands.cs.

INDICATORS

BollingerBands() 2/2

```
BollingerBands QuantConnect.Indicators.BollingerBands (
string name,
int period,
decimal k,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the BollingerBands class.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the standard deviation and moving average (middle band).
decimal	k	The number of standard deviations specifying the distance between the middle band and upper or lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used. Default: MovingAverageType.Simple.

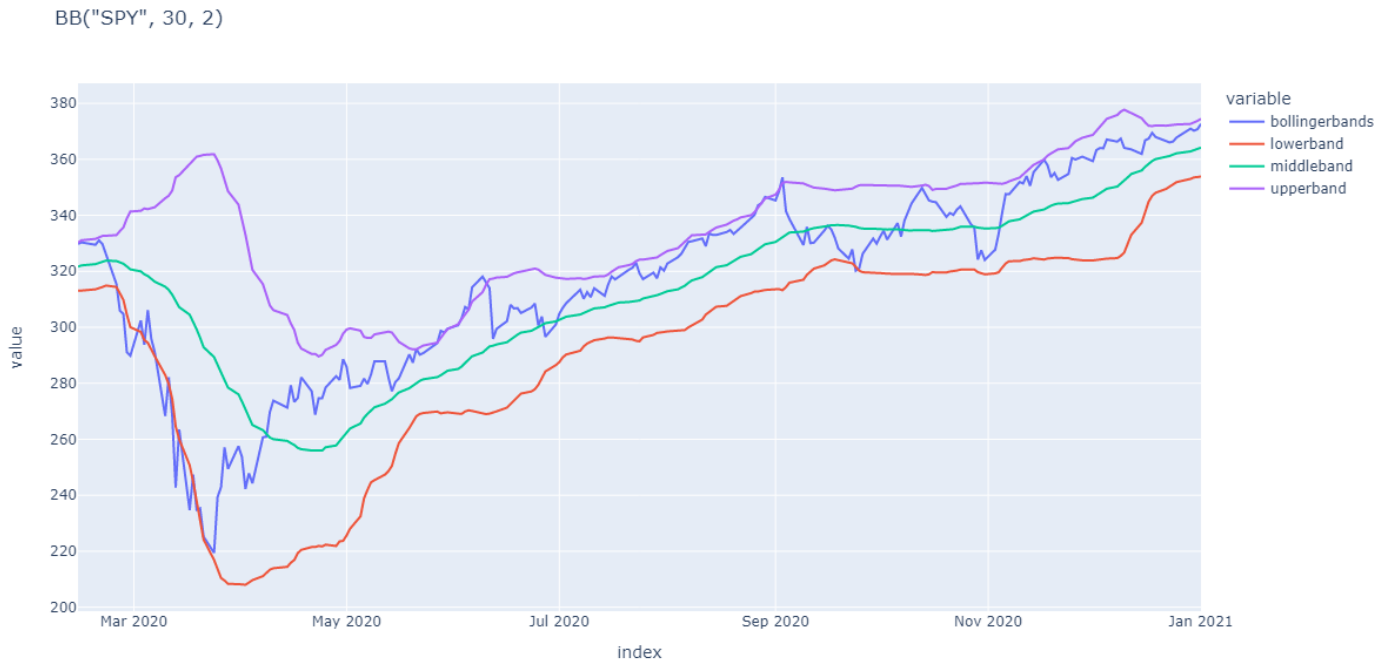
Return

`BollingerBands` - The new `BollingerBands` indicator object.

Definition at line 84 of file Indicators/BollingerBands.cs.

Visualization

The following image shows plot values of selected properties of `BollingerBands` using the plotly library.



Supported Indicators

Chaikin Money Flow

Introduction

The Chaikin Money Flow Index (CMF) is a volume-weighted average of accumulation and distribution over a specified period. CMF = n-day Sum of [(((C - L) - (H - C)) / (H - L)) x Vol] / n-day Sum of Vol Where: n = number of periods, typically 21 H = high L = low C = close Vol = volume https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/cmf

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CMF Indicator

To create an automatic indicators for `ChaikinMoneyFlow` , call the `CMF` helper method from the `QCAlgorithm` class. The `CMF` method creates a `ChaikinMoneyFlow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ChaikinMoneyFlowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cmf = self.CMF(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.cmf.IsReady:
            # The current value of self.cmf is represented by self.cmf.Current.Value
            self.Plot("ChaikinMoneyFlow", "cmf", self.cmf.Current.Value)
```

The following reference table describes the `CMF` method:

INDICATORS

CMF() 1/1

```
ChaikinMoneyFlow QuantConnect.Algorithm.QCAlgorithm.CMF (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new ChaikinMoneyFlow indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose CMF we want.
`Int32`	period	The period over which to compute the CMF.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`ChaikinMoneyFlow` - The ChaikinMoneyFlow indicator for the requested symbol over the specified period.

Definition at line 396 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ChaikinMoneyFlow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ChaikinMoneyFlowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cmf = ChaikinMoneyFlow(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.cmf.Update(bar)

        if self.cmf.IsReady:
            # The current value of self.cmf is represented by self.cmf.Current.Value
            self.Plot("ChaikinMoneyFlow", "cmf", self.cmf.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class ChaikinMoneyFlowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cmf = ChaikinMoneyFlow(self.symbol, 20)
        self.RegisterIndicator(self.symbol, self.cmf, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.cmf.IsReady:
            # The current value of self.cmf is represented by self.cmf.Current.Value
            self.Plot("ChaikinMoneyFlow", "cmf", self.cmf.Current.Value)
```

The following reference table describes the `ChaikinMoneyFlow` constructor:

INDICATORS

ChaikinMoneyFlow() 1/1

```
ChaikinMoneyFlow QuantConnect.Indicators.ChaikinMoneyFlow (
string name,
int period
)
```

Initializes a new instance of the ChaikinMoneyFlow class.

Show Details

Parameters		
`string`	name	A name for the indicator.
`int`	period	The period over which to perform computation.

Return

`ChaikinMoneyFlow` - The new `ChaikinMoneyFlow` indicator object.

Definition at line 69 of file Indicators/ChaikinMoneyFlow.cs.

Visualization

The following image shows plot values of selected properties of `ChaikinMoneyFlow` using the plotly library.


CMF("SPY", 20)
variable
chaikinmoneyflow
value
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Chande Momentum Oscillator

Introduction

This indicator computes the Chande Momentum Oscillator (CMO). CMO calculation is mostly identical to RSI. The only difference is in the last step of calculation: RSI = gain / (gain+loss) CMO = (gain-loss) / (gain+loss)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CMO Indicator

To create an automatic indicators for `ChandeMomentumOscillator` , call the `CMO` helper method from the `QCAlgorithm` class. The `CMO` method creates a `ChandeMomentumOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ChandeMomentumOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cmo = self.CMO(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.cmo.IsReady:
            # The current value of self.cmo is represented by self.cmo.Current.Value
            self.Plot("ChandeMomentumOscillator", "cmo", self.cmo.Current.Value)
```

The following reference table describes the `CMO` method:

INDICATORS

CMO() 1/1

```
ChandeMomentumOscillator QuantConnect.Algorithm.QCAlgorithm.CMO (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new ChandeMomentumOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose CMO we want.
`Int32`	period	The period over which to compute the CMO.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ChandeMomentumOscillator` - The ChandeMomentumOscillator indicator for the requested symbol over the specified period.

Definition at line 415 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ChandeMomentumOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class ChandeMomentumOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cmo = ChandeMomentumOscillator(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.cmo.Update(bar.EndTime, bar.Close)

        if self.cmo.IsReady:
            # The current value of self.cmo is represented by self.cmo.Current.Value
            self.Plot("ChandeMomentumOscillator", "cmo", self.cmo.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ChandeMomentumOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cmo = ChandeMomentumOscillator(20)
        self.RegisterIndicator(self.symbol, self.cmo, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.cmo.IsReady:
            # The current value of self.cmo is represented by self.cmo.Current.Value
            self.Plot("ChandeMomentumOscillator", "cmo", self.cmo.Current.Value)
```

The following reference table describes the `ChandeMomentumOscillator` constructor:

INDICATORS

ChandeMomentumOscillator() 1/2

```
ChandeMomentumOscillator QuantConnect.Indicators.ChandeMomentumOscillator (
int period
)
```

Initializes a new instance of the `ChandeMomentumOscillato` class using the specified period.

Show Details

Parameters		
int	period	The period of the indicator.

Return

`ChandeMomentumOscillator` - The new `ChandeMomentumOscillator` indicator object.

Definition at line 35 of file Indicators/ChandeMomentumOscillator.cs.

INDICATORS

ChandeMomentumOscillator() 2/2

```
ChandeMomentumOscillator QuantConnect.Indicators.ChandeMomentumOscillator (
string name,
int period
)
```

Initializes a new instance of the `ChandeMomentumOscillato` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the indicator.

Return

`ChandeMomentumOscillator` - The new `ChandeMomentumOscillator` indicator object.

Definition at line 45 of file Indicators/ChandeMomentumOscillator.cs.

Visualization

The following image shows plot values of selected properties of `ChandeMomentumOscillator` using the plotly library.

Supported Indicators

Commodity Channel Index

Introduction

This indicator represents the traditional commodity channel index (CCI) CCI = (Typical Price - 20-period SMA of TP) / (.015 * Mean Deviation) Typical Price (TP) = (High + Low + Close)/3 Constant = 0.015 There are four steps to calculating the Mean Deviation, first, subtract the most recent 20-period average of the typical price from each period's typical price. Second, take the absolute values of these numbers. Third, sum the absolute values. Fourth, divide by the total number of periods (20).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CCI Indicator

To create an automatic indicators for `CommodityChannelIndex` , call the `CCI` helper method from the `QCAlgorithm` class. The `CCI` method creates a `CommodityChannelIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class CommodityChannelIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cci = self.CCI(self.symbol, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.cci.IsReady:
            # The current value of self.cci is represented by self.cci.Current.Value
            self.Plot("CommodityChannelIndex", "cci", self.cci.Current.Value)
            # Plot all attributes of self.cci
            self.Plot("CommodityChannelIndex", "typicalpriceaverage",
self.cci.TypicalPriceAverage.Current.Value)
            self.Plot("CommodityChannelIndex", "typicalpricemeandeviation",
self.cci.TypicalPriceMeanDeviation.Current.Value)
```

The following reference table describes the `CCI` method:

INDICATORS

CCI() 1/1

```
CommodityChannelIndex QuantConnect.Algorithm.QCAlgorithm.CCI (
Symbol symbol,
Int32 period,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new CommodityChannelIndex indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose CCI we want.
`Int32`	period	The period over which to compute the CCI.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use in computing the typical price average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`CommodityChannelIndex` - The CommodityChannelIndex indicator for the requested symbol over the specified period.

Definition at line 378 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `CommodityChannelIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class CommodityChannelIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cci = CommodityChannelIndex(20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.cci.Update(bar)

        if self.cci.IsReady:
            # The current value of self.cci is represented by self.cci.Current.Value
            self.Plot("CommodityChannelIndex", "cci", self.cci.Current.Value)
            # Plot all attributes of self.cci
            self.Plot("CommodityChannelIndex", "typicalpriceaverage",
self.cci.TypicalPriceAverage.Current.Value)
            self.Plot("CommodityChannelIndex", "typicalpricemeandeviation",
self.cci.TypicalPriceMeanDeviation.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class CommodityChannelIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cci = CommodityChannelIndex(20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.cci, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.cci.IsReady:
            # The current value of self.cci is represented by self.cci.Current.Value
            self.Plot("CommodityChannelIndex", "cci", self.cci.Current.Value)
            # Plot all attributes of self.cci
            self.Plot("CommodityChannelIndex", "typicalpriceaverage",
self.cci.TypicalPriceAverage.Current.Value)
            self.Plot("CommodityChannelIndex", "typicalpricemeandeviation",
self.cci.TypicalPriceMeanDeviation.Current.Value)
```

The following reference table describes the `CommodityChannelIndex` constructor:

INDICATORS

CommodityChannelIndex() 1/2

```
CommodityChannelIndex QuantConnect.Indicators.CommodityChannelIndex (
int period,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the CommodityChannelIndex class.

Show Details

Parameters		
int	period	The period of the standard deviation and moving average (middle band).
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used. Default: MovingAverageType.Simple.

Return

CommodityChannelIndex - The new CommodityChannelIndex indicator object.

Definition at line 59 of file Indicators/CommodityChannelIndex.cs.

INDICATORS

CommodityChannelIndex() 2/2

```
CommodityChannelIndex QuantConnect.Indicators.CommodityChannelIndex (
string name,
int period,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the CommodityChannelIndex class.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the standard deviation and moving average (middle band).
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used. Default: MovingAverageType.Simple.

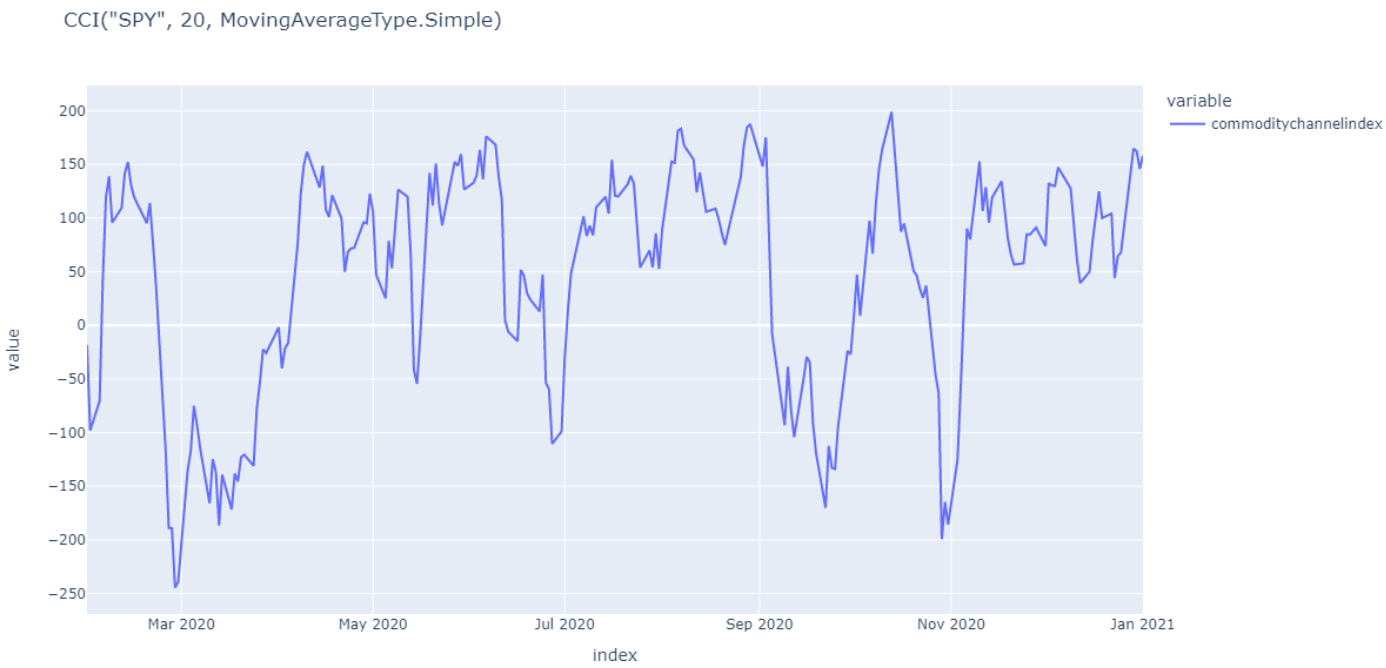
Return

CommodityChannelIndex - The new CommodityChannelIndex indicator object.

Definition at line 70 of file Indicators/CommodityChannelIndex.cs.

Visualization

The following image shows plot values of selected properties of `CommodityChannelIndex` using the plotly library.



Supported Indicators

Coppock Curve

Introduction

A momentum indicator developed by Edwin “Sedge” Coppock in October 1965. The goal of this indicator is to identify long-term buying opportunities in the S&P500 and Dow Industrials. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CC Indicator

To create an automatic indicators for `CoppockCurve` , call the `CC` helper method from the `QCAlgorithm` class. The `CC` method creates a `CoppockCurve` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class CoppockCurveAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cc = self.CC(self.symbol, 11, 14, 10)

    def OnData(self, slice: Slice) -> None:
        if self.cc.IsReady:
            # The current value of self.cc is represented by self.cc.Current.Value
            self.Plot("CoppockCurve", "cc", self.cc.Current.Value)
```

The following reference table describes the `CC` method:

INDICATORS

CC() 1/1

```
CoppockCurve QuantConnect.Algorithm.QCAlgorithm.CC (
Symbol symbol,
*Int32 shortRocPeriod,
*Int32 longRocPeriod,
*Int32 lwmaPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Initializes a new instance of the CoppockCurve" indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Coppock Curve we want.
`*Int32`	shortRocPeriod	*(Optional)* The period for the short ROC.
`*Int32`	longRocPeriod	*(Optional)* The period for the long ROC.
`*Int32`	lwmaPeriod	*(Optional)* The period for the LWMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`CoppockCurve` - The Coppock Curve indicator for the requested symbol over the specified period.

Definition at line 357 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `CoppockCurve` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class CoppockCurveAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cc = CoppockCurve(11, 14, 10)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.cc.Update(bar.EndTime, bar.Close)

        if self.cc.IsReady:
            # The current value of self.cc is represented by self.cc.Current.Value
            self.Plot("CoppockCurve", "cc", self.cc.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class CoppockCurveAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.cc = CoppockCurve(11, 14, 10)
        self.RegisterIndicator(self.symbol, self.cc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.cc.IsReady:
            # The current value of self.cc is represented by self.cc.Current.Value
            self.Plot("CoppockCurve", "cc", self.cc.Current.Value)
```

The following reference table describes the `CoppockCurve` constructor:

INDICATORS

CoppockCurve() 1/3

```
CoppockCurve QuantConnect.Indicators.CoppockCurve (

)
```

Initializes a new instance of the `CoppockCurve` indicator with its default values.

Show Details

This method requires no argument input.

Return

`CoppockCurve` - The new `CoppockCurve` indicator object.

Definition at line 44 of file Indicators/CoppockCurve.cs.

INDICATORS

CoppockCurve() 2/3

```
CoppockCurve QuantConnect.Indicators.CoppockCurve (
 int shortRocPeriod,
 int longRocPeriod,
 int lwmaPeriod
)
```

Initializes a new instance of the `CoppockCurv` indicator.

Show Details

Parameters		
`int`	shortRocPeriod	The period for the short ROC.
`int`	longRocPeriod	The period for the long ROC.
`int`	lwmaPeriod	The period for the LWMA.

Return

`CoppockCurve` - The new `CoppockCurve` indicator object.

Definition at line 55 of file Indicators/CoppockCurve.cs.

INDICATORS

CoppockCurve() 3/3

```
CoppockCurve QuantConnect.Indicators.CoppockCurve (
 string name,
 int shortRocPeriod,
 int longRocPeriod,
 int lwmaPeriod
)
```

Initializes a new instance of the `CoppockCurve` indicator.

Show Details

Parameters		
`string`	name	A name for the indicator.
`int`	shortRocPeriod	The period for the short ROC.
`int`	longRocPeriod	The period for the long ROC.
`int`	lwmaPeriod	The period for the LWMA.

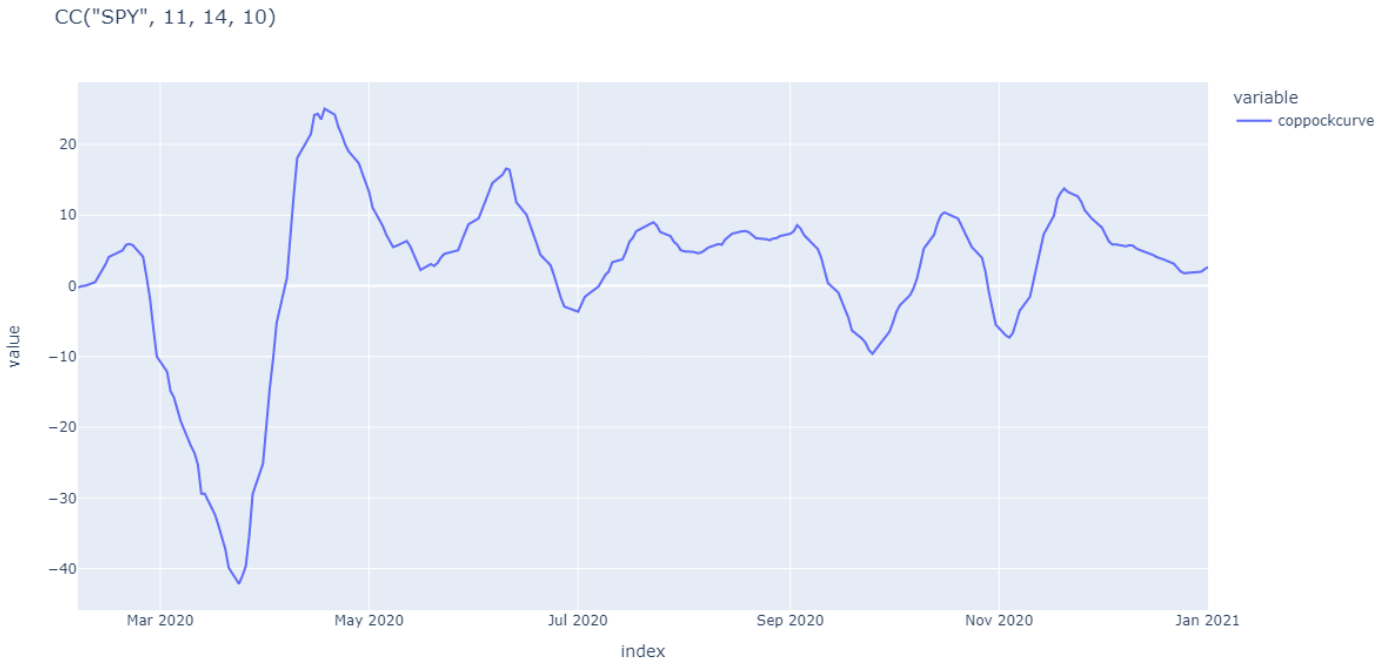
Return

`CoppockCurve` - The new `CoppockCurve` indicator object.

Definition at line 67 of file Indicators/CoppockCurve.cs.

Visualization

The following image shows plot values of selected properties of `CoppockCurve` using the plotly library.



Supported Indicators

De Marker Indicator

Introduction

In the DeMarker strategy, for some period of size N, set: DeMax = High - Previous High, and DeMin = Previous Low - Low where, in the prior, if either term is less than zero (DeMax or DeMin), set it to zero. We can now define the indicator itself, DEM, as: DEM = MA(DeMax)/(MA(DeMax)+MA(DeMin)) where MA denotes a Moving Average of period N. https://www.investopedia.com/terms/d/demarkerindicator.asp

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DEM Indicator

To create an automatic indicators for `DeMarkerIndicator` , call the `DEM` helper method from the `QCAlgorithm` class. The `DEM` method creates a `DeMarkerIndicator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class DeMarkerIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dem = self.DEM(self.symbol, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.dem.IsReady:
            # The current value of self.dem is represented by self.dem.Current.Value
            self.Plot("DeMarkerIndicator", "dem", self.dem.Current.Value)
```

The following reference table describes the `DEM` method:

INDICATORS

DEM() 1/1

```
DeMarkerIndicator QuantConnect.Algorithm.QCAlgorithm.DEM (
Symbol symbol,
Int32 period,
MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new DeMarker Indicator (DEM), an oscillator-type indicator measuring changes in terms of an asset's High and Low tradebar values.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose DEM we seek.
`Int32`	period	The period of the moving average implemented.
`MovingAverageType`	type	Specifies the type of moving average to be used.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`DeMarkerIndicator` - The DeMarker indicator for the requested symbol.

Definition at line 435 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `DeMarkerIndicator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class DeMarkerIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dem = DeMarkerIndicator(20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.dem.Update(bar)

        if self.dem.IsReady:
            # The current value of self.dem is represented by self.dem.Current.Value
            self.Plot("DeMarkerIndicator", "dem", self.dem.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class DeMarkerIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dem = DeMarkerIndicator(20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.dem, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.dem.IsReady:
            # The current value of self.dem is represented by self.dem.Current.Value
            self.Plot("DeMarkerIndicator", "dem", self.dem.Current.Value)
```

The following reference table describes the `DeMarkerIndicator` constructor:

INDICATORS

DeMarkerIndicator() 1/2

```
DeMarkerIndicator QuantConnect.Indicators.DeMarkerIndicator (
int period,
*MovingAverageType type
)
```

Initializes a new instance of the DeMarkerIndicator class with the specified period.

Show Details

Parameters		
`int`	period	The period of the DeMarker Indicator.
`*MovingAverageType`	type	*(Optional) (Optional)* The type of moving average to use in calculations. Default: MovingAverageType.Simple.

Return

`DeMarkerIndicator` - The new `DeMarkerIndicator` indicator object.

Definition at line 48 of file Indicators/DeMarkerIndicator.cs.

INDICATORS

DeMarkerIndicator() 2/2

```
DeMarkerIndicator QuantConnect.Indicators.DeMarkerIndicator (
string name,
int period,
*MovingAverageType type
)
```

Initializes a new instance of the DeMarkerIndicator class with the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the DeMarker Indicator.
`*MovingAverageType`	type	*(Optional) (Optional)* The type of moving average to use in calculations. Default: MovingAverageType.Simple.

Return

`DeMarkerIndicator` - The new `DeMarkerIndicator` indicator object.

Definition at line 59 of file Indicators/DeMarkerIndicator.cs.

Visualization

The following image shows plot values of selected properties of `DeMarkerIndicator` using the plotly library.

DEM("SPY", 20, MovingAverageType.Simple)

Supported Indicators

Delay

Introduction

An indicator that delays its input for a certain period

To view the implementation of this indicator, see the LEAN GitHub repository .

Using Delay Indicator

Delay does not have an automatic indicator implementation available.

You can manually create a `Delay` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```
PY
class DelayAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.delay = Delay()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.delay.Update(bar.EndTime, bar.Close)

        if self.delay.IsReady:
            # The current value of self.delay is represented by self.delay.Current.Value
            self.Plot("Delay", "delay", self.delay.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
PY
class DelayAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.delay = Delay()
        self.RegisterIndicator(self.symbol, self.delay, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.delay.IsReady:
            # The current value of self.delay is represented by self.delay.Current.Value
            self.Plot("Delay", "delay", self.delay.Current.Value)
```

The following reference table describes the `Delay` constructor:

INDICATORS

Delay() 1/2

```
Delay QuantConnect.Indicators.Delay (
int period,
)
```

Initializes a new instance of the Delay class.

Show Details

Parameters		
int	period	The period to delay input, must be greater than zero.

Return

`Delay` - The new `Delay` indicator object.

Definition at line 27 of file Indicators/Delay.cs.

INDICATORS

Delay() 2/2

```
Delay QuantConnect.Indicators.Delay (
string name,
int period,
)
```

Initializes a new instance of the Delay class.

Show Details

Parameters		
string	name	Name of the delay window indicator.
int	period	The period to delay input, must be greater than zero.

Return

`Delay` - The new `Delay` indicator object.

Definition at line 37 of file Indicators/Delay.cs.

Visualization

The following image shows plot values of selected properties of `Delay` using the plotly library.

Supported Indicators

Detrended Price Oscillator

Introduction

The Detrended Price Oscillator is an indicator designed to remove trend from price and make it easier to identify cycles. DPO does not extend to the last date because it is based on a displaced moving average. Is estimated as Price {X/2 + 1} periods ago less the X-period simple moving average. E.g.DPO(20) equals price 11 days ago less the 20-day SMA.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DPO Indicator

To create an automatic indicators for `DetrendedPriceOscillator` , call the `DPO` helper method from the `QCAlgorithm` class. The `DPO` method creates a `DetrendedPriceOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class DetrendedPriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dpo = self.DPO(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.dpo.IsReady:
            # The current value of self.dpo is represented by self.dpo.Current.Value
            self.Plot("DetrendedPriceOscillator", "dpo", self.dpo.Current.Value)
```

The following reference table describes the `DPO` method:

INDICATORS

DPO() 1/1

```
DetrendedPriceOscillator QuantConnect.Algorithm.QCAlgorithm.DPO (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new DetrendedPriceOscillator" indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose DPO we want.
`Int32`	period	The period over which to compute the DPO.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`DetrendedPriceOscillator` - A new registered DetrendedPriceOscillator indicator for the requested symbol over the specified period.

Definition at line 505 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `DetrendedPriceOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class DetrendedPriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dpo = DetrendedPriceOscillator(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.dpo.Update(bar.EndTime, bar.Close)

        if self.dpo.IsReady:
            # The current value of self.dpo is represented by self.dpo.Current.Value
            self.Plot("DetrendedPriceOscillator", "dpo", self.dpo.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class DetrendedPriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dpo = DetrendedPriceOscillator(20)
        self.RegisterIndicator(self.symbol, self.dpo, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.dpo.IsReady:
            # The current value of self.dpo is represented by self.dpo.Current.Value
            self.Plot("DetrendedPriceOscillator", "dpo", self.dpo.Current.Value)
```

The following reference table describes the `DetrendedPriceOscillator` constructor:

INDICATORS

DetrendedPriceOscillator() 1/2

```
DetrendedPriceOscillator QuantConnect.Indicators.DetrendedPriceOscillator (
string name,
int period
)
```

Initializes a new instance of the `DetrendedPriceOscillator` class.

Show Details

Parameters		
`string`	name	The name for the indicator.
`int`	period	The number of periods to calculate the DPO.

Return

`DetrendedPriceOscillator` - The new `DetrendedPriceOscillator` indicator object.

Definition at line 46 of file Indicators/DetrendedPriceOscillator.cs.

INDICATORS

DetrendedPriceOscillator() 2/2

```
DetrendedPriceOscillator QuantConnect.Indicators.DetrendedPriceOscillator (
int period
)
```

Initializes a new instance of the `DetrendedPriceOscillator` class.

Show Details

Parameters		
`int`	period	The number of periods to calculate the DPO.

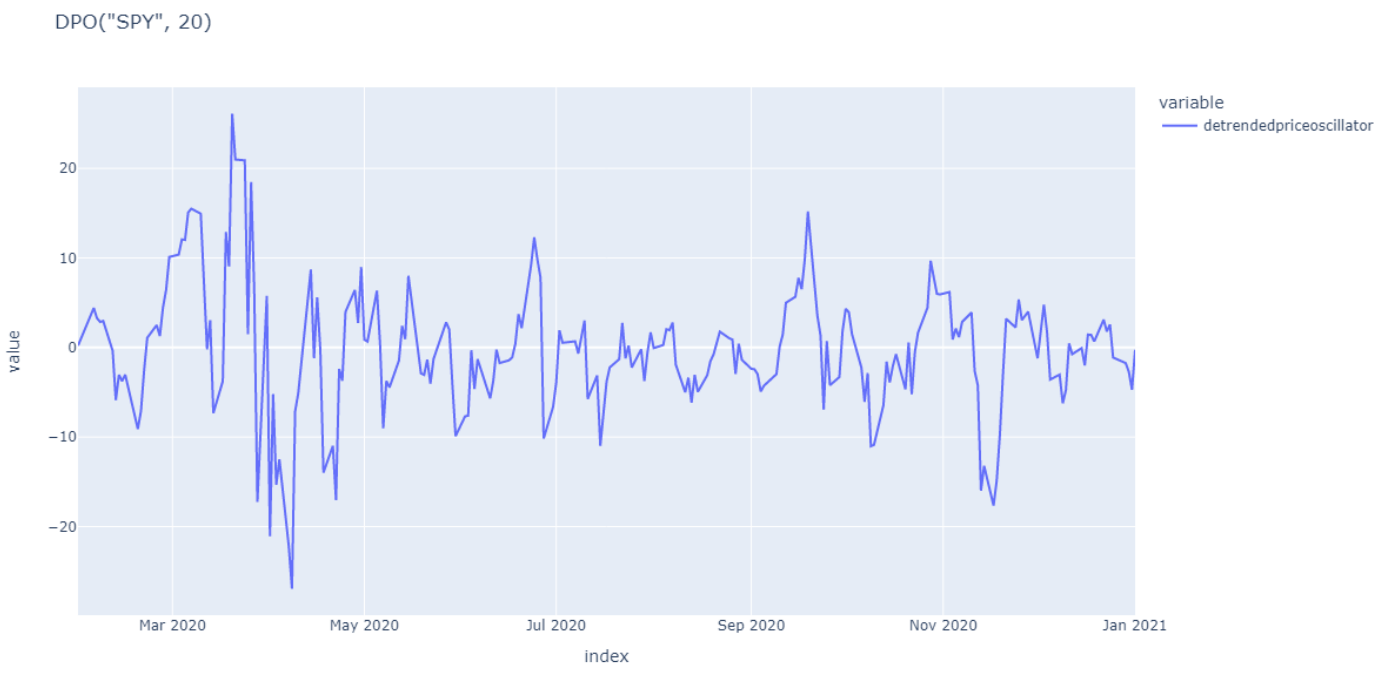
Return

`DetrendedPriceOscillator` - The new `DetrendedPriceOscillator` indicator object.

Definition at line 59 of file Indicators/DetrendedPriceOscillator.cs.

Visualization

The following image shows plot values of selected properties of `DetrendedPriceOscillator` using the plotly library.



Supported Indicators

Donchian Channel

Introduction

This indicator computes the upper and lower band of the Donchian Channel. The upper band is computed by finding the highest high over the given period. The lower band is computed by finding the lowest low over the given period. The primary output value of the indicator is the mean of the upper and lower band for the given timeframe.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DCH Indicator

To create an automatic indicators for `DonchianChannel` , call the `DCH` helper method from the `QCAlgorithm` class. The `DCH` method creates a `DonchianChannel` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class DonchianChannelAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dch = self.DCH(self.symbol, 20, 20)

    def OnData(self, slice: Slice) -> None:
        if self.dch.IsReady:
            # The current value of self.dch is represented by self.dch.Current.Value
            self.Plot("DonchianChannel", "dch", self.dch.Current.Value)
            # Plot all attributes of self.dch
            self.Plot("DonchianChannel", "upperband", self.dch.UpperBand.Current.Value)
            self.Plot("DonchianChannel", "lowerband", self.dch.LowerBand.Current.Value)
```

The following reference table describes the `DCH` method:

INDICATORS

DCH() 1/2

```
DonchianChannel QuantConnect.Algorithm.QCAlgorithm.DCH (
 Symbol symbol,
 Int32 upperPeriod,
 Int32 lowerPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Donchian Channel indicator which will compute the Upper Band and Lower Band. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Donchian Channel we seek.
`Int32`	upperPeriod	The period over which to compute the upper Donchian Channel.
`Int32`	lowerPeriod	The period over which to compute the lower Donchian Channel.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`DonchianChannel` - The Donchian Channel indicator for the requested symbol.

Definition at line 454 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

DCH() 2/2

```
DonchianChannel QuantConnect.Algorithm.QCAlgorithm.DCH (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
)
```

Overload shorthand to create a new symmetric Donchian Channel indicator which has the upper and lower channels set to the same period length.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Donchian Channel we seek.
`Int32`	period	The period over which to compute the Donchian Channel.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`DonchianChannel` - The Donchian Channel indicator for the requested symbol.

Definition at line 473 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `DonchianChannel` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class DonchianChannelAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dch = DonchianChannel(20, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.dch.Update(bar)

        if self.dch.IsReady:
            # The current value of self.dch is represented by self.dch.Current.Value
            self.Plot("DonchianChannel", "dch", self.dch.Current.Value)
            # Plot all attributes of self.dch
            self.Plot("DonchianChannel", "upperband", self.dch.UpperBand.Current.Value)
            self.Plot("DonchianChannel", "lowerband", self.dch.LowerBand.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class DonchianChannelAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dch = DonchianChannel(20, 20)
        self.RegisterIndicator(self.symbol, self.dch, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.dch.IsReady:
            # The current value of self.dch is represented by self.dch.Current.Value
            self.Plot("DonchianChannel", "dch", self.dch.Current.Value)
            # Plot all attributes of self.dch
            self.Plot("DonchianChannel", "upperband", self.dch.UpperBand.Current.Value)
            self.Plot("DonchianChannel", "lowerband", self.dch.LowerBand.Current.Value)
```

The following reference table describes the `DonchianChannel` constructor:

INDICATORS

DonchianChannel() 1/4

```
DonchianChannel QuantConnect.Indicators.DonchianChannel (
int period
)
```

Initializes a new instance of the `DonchianChanne` class.

Show Details

Parameters		
`int`	period	The period for both the upper and lower channels.

Return

`DonchianChannel` - The new `DonchianChannel` indicator object.

Definition at line 46 of file Indicators/DonchianChannel.cs.

INDICATORS

DonchianChannel() 2/4

```
DonchianChannel QuantConnect.Indicators.DonchianChannel (
int upperPeriod,
int lowerPeriod
)
```

Initializes a new instance of the `DonchianChanne` class.

Show Details

Parameters		
`int`	upperPeriod	The period for the upper channel.
`int`	lowerPeriod	The period for the lower channel.

Return

`DonchianChannel` - The new `DonchianChannel` indicator object.

Definition at line 56 of file Indicators/DonchianChannel.cs.

INDICATORS

DonchianChannel() 3/4

```
DonchianChannel QuantConnect.Indicators.DonchianChannel (
 string name,
 int period
 )
```

Initializes a new instance of the `DonchianChanne` class.

Show Details

Parameters		
`string`	name	The name.
`int`	period	The period for both the upper and lower channels.

Return

`DonchianChannel` - The new `DonchianChannel` indicator object.

Definition at line 66 of file Indicators/DonchianChannel.cs.

INDICATORS

DonchianChannel() 4/4

```
DonchianChannel QuantConnect.Indicators.DonchianChannel (
string name,
int upperPeriod,
int lowerPeriod
)
```

Initializes a new instance of the `DonchianChanne` class.

Show Details

Parameters		
`string`	name	The name.
`int`	upperPeriod	The period for the upper channel.
`int`	lowerPeriod	The period for the lower channel.

Return

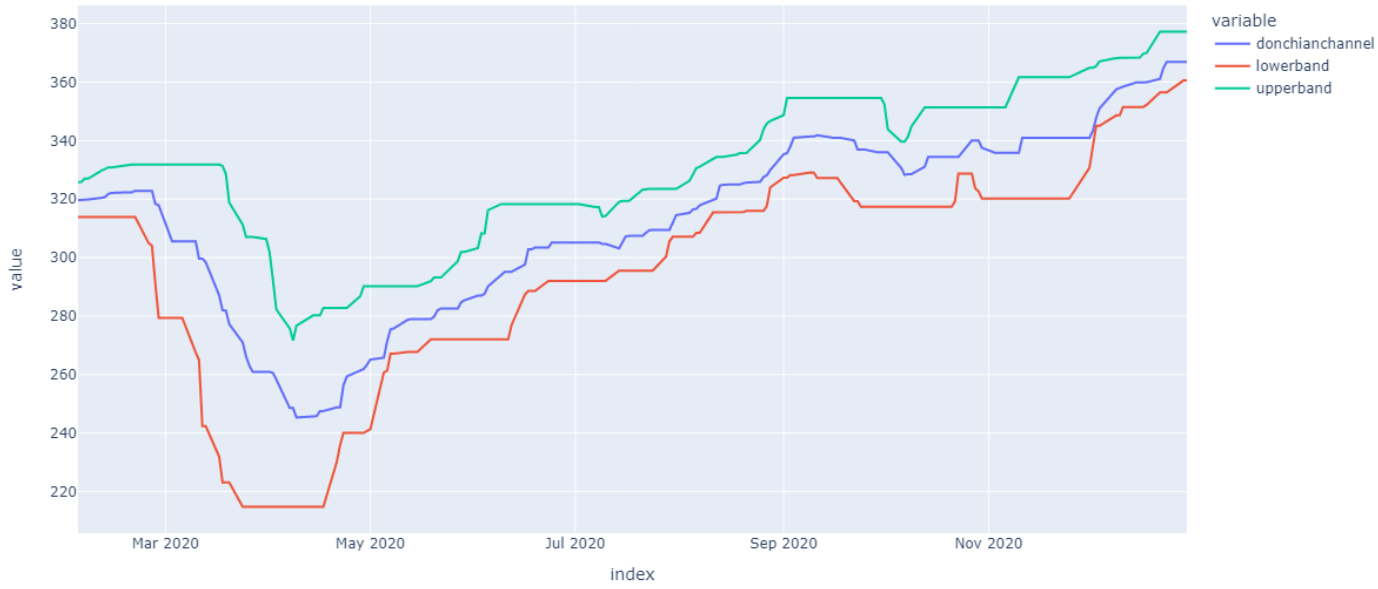
`DonchianChannel` - The new `DonchianChannel` indicator object.

Definition at line 77 of file Indicators/DonchianChannel.cs.

Visualization

The following image shows plot values of selected properties of `DonchianChannel` using the plotly library.

DCH("SPY", 20, 20)


variable
donchianchannel
lowerband
upperband
value
380
360
340
320
300
280
260
240
220
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
index

Supported Indicators

Double Exponential Moving Average

Introduction

This indicator computes the Double Exponential Moving Average (DEMA). The Double Exponential Moving Average is calculated with the following formula: EMA2 = EMA(EMA(t,period),period) DEMA = 2 * EMA(t,period) - EMA2 The Generalized DEMA (GD) is calculated with the following formula: GD = (volumeFactor+1) * EMA(t,period) - volumeFactor * EMA2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DEMA Indicator

To create an automatic indicators for `DoubleExponentialMovingAverage` , call the `DEMA` helper method from the `QCAlgorithm` class. The `DEMA` method creates a `DoubleExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class DoubleExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dema = self.DEMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.dema.IsReady:
            # The current value of self.dema is represented by self.dema.Current.Value
            self.Plot("DoubleExponentialMovingAverage", "dema", self.dema.Current.Value)
```

The following reference table describes the `DEMA` method:

INDICATORS

DEMA() 1/1

```
DoubleExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.DEMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new DoubleExponentialMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose DEMA we want.
`Int32`	period	The period over which to compute the DEMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`DoubleExponentialMovingAverage` - The DoubleExponentialMovingAverage indicator for the requested symbol over the specified period.

Definition at line 487 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `DoubleExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class DoubleExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dema = DoubleExponentialMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.dema.Update(bar.EndTime, bar.Close)

        if self.dema.IsReady:
            # The current value of self.dema is represented by self.dema.Current.Value
            self.Plot("DoubleExponentialMovingAverage", "dema", self.dema.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class DoubleExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.dema = DoubleExponentialMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.dema, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.dema.IsReady:
            # The current value of self.dema is represented by self.dema.Current.Value
            self.Plot("DoubleExponentialMovingAverage", "dema", self.dema.Current.Value)
```

The following reference table describes the `DoubleExponentialMovingAverage` constructor:

INDICATORS

DoubleExponentialMovingAverage() 1/2

```
DoubleExponentialMovingAverage QuantConnect.Indicators.DoubleExponentialMovingAverage (
string name,
int period,
*decimal volumeFactor
)
```

Initializes a new instance of the `DoubleExponentialMovingAverag` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the DEMA.
`*decimal`	volumeFactor	*(Optional) (Optional)* The volume factor of the DEMA (value must be in the [0,1] range, set to 1 for standard DEMA). Default: 1m.

Return

`DoubleExponentialMovingAverage` - The new `DoubleExponentialMovingAverage` indicator object.

Definition at line 39 of file Indicators/DoubleExponentialMovingAverage.cs.

INDICATORS

DoubleExponentialMovingAverage() 2/2

```
DoubleExponentialMovingAverage QuantConnect.Indicators.DoubleExponentialMovingAverage (
int period,
*decimal volumeFactor
)
```

Initializes a new instance of the `DoubleExponentialMovingAverag` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the DEMA.
`*decimal`	volumeFactor	*(Optional) (Optional)* The volume factor of the DEMA (value must be in the [0,1] range, set to 1 for standard DEMA). Default: 1m.

Return

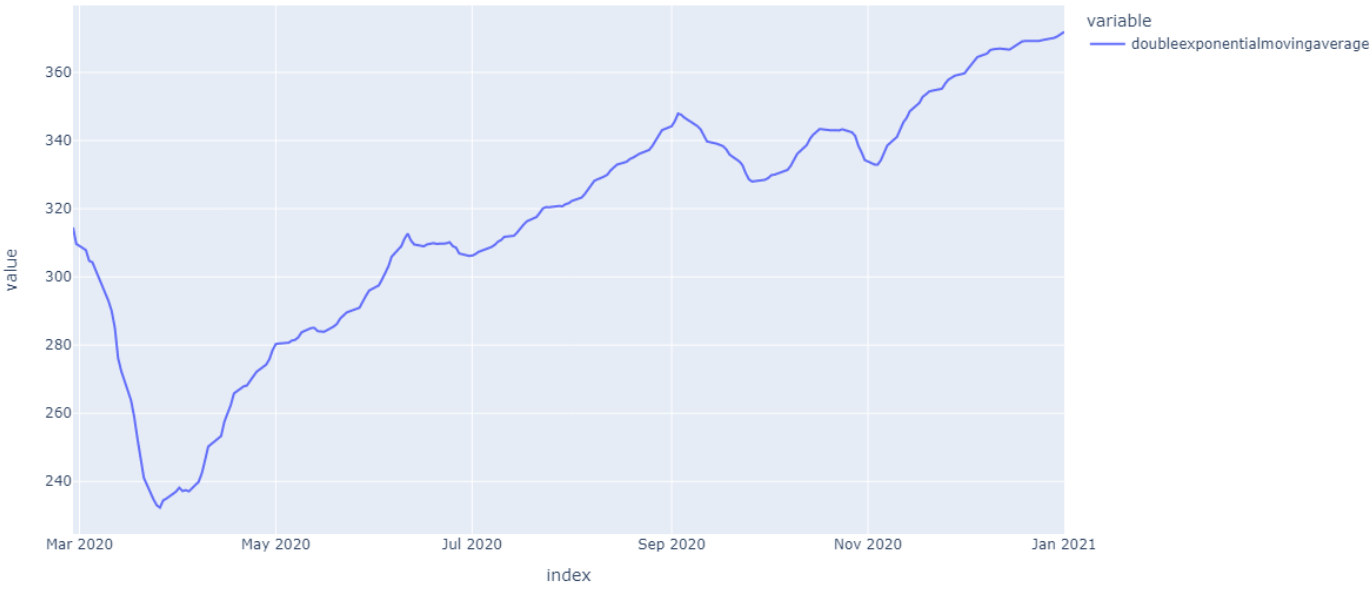
`DoubleExponentialMovingAverage` - The new `DoubleExponentialMovingAverage` indicator object.

Definition at line 53 of file Indicators/DoubleExponentialMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `DoubleExponentialMovingAverage` using the plotly library.

DEMA("SPY", 20)


variable
doubleexponentialmovingaverage
value
360
340
320
300
280
260
240
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Ease Of Movement Value

Introduction

This indicator computes the n-period Ease of Movement Value using the following: MID = (high_1 + low_1)/2 - (high_0 + low_0)/2 RATIO = (currentVolume/10000) / (high_1 - low_1) EMV = MID/RATIO _SMA = n-period of EMV Returns _SMA source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using EMV Indicator

To create an automatic indicators for `EaseOfMovementValue` , call the `EMV` helper method from the `QCAlgorithm` class. The `EMV` method creates a `EaseOfMovementValue` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class EaseOfMovementValueAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.emv = self.EMV(self.symbol, 1, 10000)

    def OnData(self, slice: Slice) -> None:
        if self.emv.IsReady:
            # The current value of self.emv is represented by self.emv.Current.Value
            self.Plot("EaseOfMovementValue", "emv", self.emv.Current.Value)
```

The following reference table describes the `EMV` method:

INDICATORS

EMV() 1/1

```
EaseOfMovementValue QuantConnect.Algorithm.QCAlgorithm.EMV (
Symbol symbol,
*Int32 period,
*Int32 scale,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an EaseOfMovementValue indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EMV we want.
`*Int32`	period	*(Optional)* The period of the EMV.
`*Int32`	scale	*(Optional)* The length of the outputed value.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`EaseOfMovementValue` - The EaseOfMovementValue indicator for the given parameters.

Definition at line 560 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `EaseOfMovementValue` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class EaseOfMovementValueAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.emv = EaseOfMovementValue(1, 10000)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.emv.Update(bar)

        if self.emv.IsReady:
            # The current value of self.emv is represented by self.emv.Current.Value
            self.Plot("EaseOfMovementValue", "emv", self.emv.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class EaseOfMovementValueAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.emv = EaseOfMovementValue(1, 10000)
        self.RegisterIndicator(self.symbol, self.emv, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.emv.IsReady:
            # The current value of self.emv is represented by self.emv.Current.Value
            self.Plot("EaseOfMovementValue", "emv", self.emv.Current.Value)
```

The following reference table describes the `EaseOfMovementValue` constructor:

INDICATORS

EaseOfMovementValue() 1/2

```
EaseOfMovementValue QuantConnect.Indicators.EaseOfMovementValue (
*int period,
*int scale
)
```

Initializeds a new instance of the EaseOfMovement class using the specufued period.

Show Details

Parameters		
*int	period	*(Optional) (Optional)* The period over which to perform to computation. Default: 1.
*int	scale	*(Optional) (Optional)* The size of the number outputed by EMV. Default: 10000.

Return

`EaseOfMovementValue` - The new `EaseOfMovementValue` indicator object.

Definition at line 50 of file Indicators/EaseOfMovementValue.cs.

INDICATORS

EaseOfMovementValue() 2/2

```
EaseOfMovementValue QuantConnect.Indicators.EaseOfMovementValue (
string name,
int period,
int scale
)
```

Creates a new EaseOfMovement indicator with the specified period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period over which to perform to computation.
int	scale	The size of the number outputed by EMV.

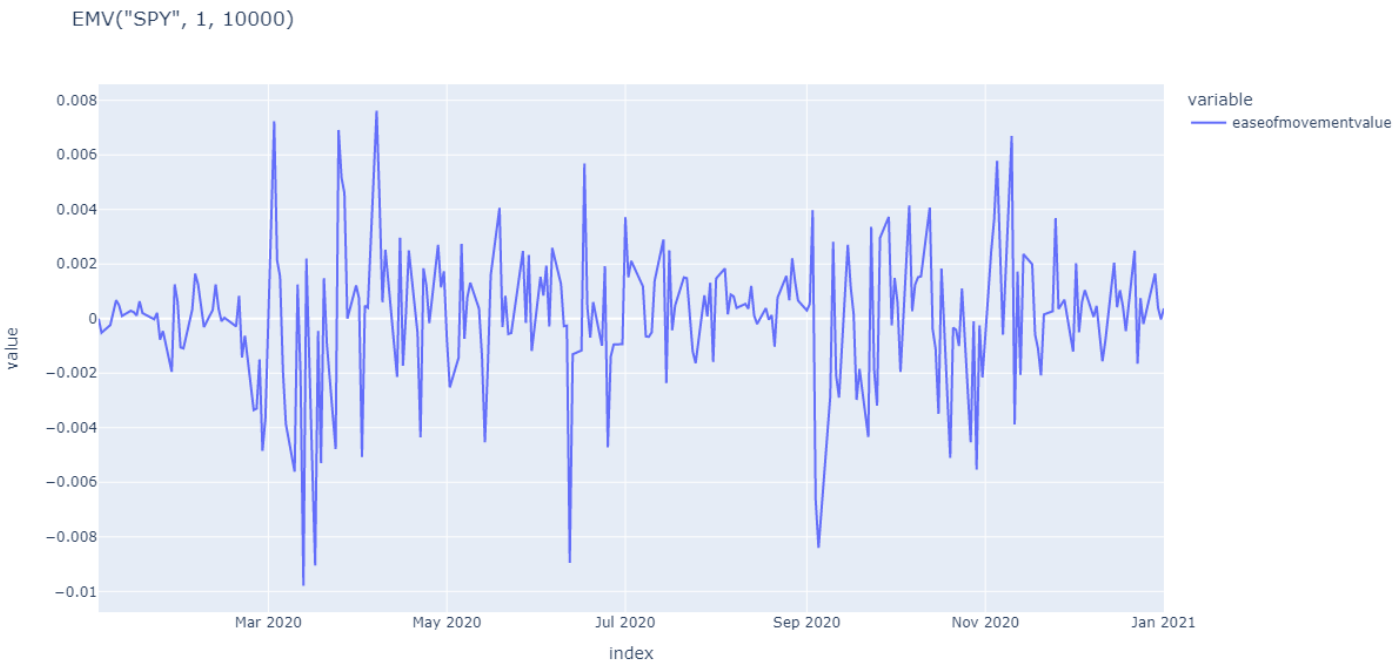
Return

`EaseOfMovementValue` - The new `EaseOfMovementValue` indicator object.

Definition at line 60 of file Indicators/EaseOfMovementValue.cs.

Visualization

The following image shows plot values of selected properties of `EaseOfMovementValue` using the plotly library.



Supported Indicators

Exponential Moving Average

Introduction

This indicator represents the traditional exponential moving average indicator (EMA). When the indicator is ready, the first value of the EMA is equivalent to the simple moving average. After the first EMA value, the EMA value is a function of the previous EMA value. Therefore, depending on the number of samples you feed into the indicator, it can provide different EMA values for a single security and lookback period. To make the indicator values consistent across time, warm up the indicator with all the trailing security price history.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using EMA Indicator

To create an automatic indicators for `ExponentialMovingAverage` , call the `EMA` helper method from the `QCAlgorithm` class. The `EMA` method creates a `ExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class ExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ema = self.EMA(self.symbol, 20, 0.5)

    def OnData(self, slice: Slice) -> None:
        if self.ema.IsReady:
            # The current value of self.ema is represented by self.ema.Current.Value
            self.Plot("ExponentialMovingAverage", "ema", self.ema.Current.Value)
```

The following reference table describes the `EMA` method:

INDICATORS

EMA() 1/2

```
ExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.EMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an ExponentialMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EMA we want.
`Int32`	period	The period of the EMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ExponentialMovingAverage` - The ExponentialMovingAverage for the given parameters.

Definition at line 524 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

EMA() 2/2

```
ExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.EMA (
 Symbol symbol,
 Int32 period,
 Decimal smoothingFactor,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates an ExponentialMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EMA we want.
`Int32`	period	The period of the EMA.
`Decimal`	smoothingFactor	The percentage of data from the previous value to be carried into the next value.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ExponentialMovingAverage` - The ExponentialMovingAverage for the given parameters.

Definition at line 540 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class ExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ema = ExponentialMovingAverage(20, 0.5)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ema.Update(bar.EndTime, bar.Close)

        if self.ema.IsReady:
            # The current value of self.ema is represented by self.ema.Current.Value
            self.Plot("ExponentialMovingAverage", "ema", self.ema.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ema = ExponentialMovingAverage(20, 0.5)
        self.RegisterIndicator(self.symbol, self.ema, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ema.IsReady:
            # The current value of self.ema is represented by self.ema.Current.Value
            self.Plot("ExponentialMovingAverage", "ema", self.ema.Current.Value)
```

The following reference table describes the `ExponentialMovingAverage` constructor:

INDICATORS

ExponentialMovingAverage() 1/4

```
ExponentialMovingAverage QuantConnect.Indicators.ExponentialMovingAverage (
string name,
int period
)
```

Initializes a new instance of the ExponentialMovingAverage class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the EMA.

Return

`ExponentialMovingAverage` - The new `ExponentialMovingAverage` indicator object.

Definition at line 44 of file Indicators/ExponentialMovingAverage.cs.

INDICATORS

ExponentialMovingAverage() 2/4

```
ExponentialMovingAverage QuantConnect.Indicators.ExponentialMovingAverage (
string name,
int period,
decimal smoothingFactor
)
```

Initializes a new instance of the ExponentialMovingAverage class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the EMA.
decimal	smoothingFactor	The percentage of data from the previous value to be carried into the next value.

Return

`ExponentialMovingAverage` - The new `ExponentialMovingAverage` indicator object.

Definition at line 55 of file Indicators/ExponentialMovingAverage.cs.

INDICATORS

ExponentialMovingAverage() 3/4

```
ExponentialMovingAverage QuantConnect.Indicators.ExponentialMovingAverage (
int period
)
```

Initializes a new instance of the ExponentialMovingAverage class with the default name and period.

Show Details

Parameters		
int	period	The period of the EMA.

Return

`ExponentialMovingAverage` - The new `ExponentialMovingAverage` indicator object.

Definition at line 67 of file Indicators/ExponentialMovingAverage.cs.

INDICATORS

ExponentialMovingAverage() 4/4

```
ExponentialMovingAverage QuantConnect.Indicators.ExponentialMovingAverage (
int period,
decimal smoothingFactor
)
```

Initializes a new instance of the ExponentialMovingAverage class with the default name and period.

Show Details

Parameters		
`int`	period	The period of the EMA.
`decimal`	smoothingFactor	The percentage of data from the previous value to be carried into the next value.

Return

`ExponentialMovingAverage` - The new `ExponentialMovingAverage` indicator object.

Definition at line 77 of file Indicators/ExponentialMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `ExponentialMovingAverage` using the plotly library.

EMA("SPY", 20, 0.5)


variable
exponentialmovingaverage
value
360
340
320
300
280
260
240
220
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Filtered Identity

Introduction

This indicator represents an indicator that is a ready after ingesting a single sample and always returns the same value as it is given if it passes a filter condition

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FilteredIdentity Indicator

To create an automatic indicators for `FilteredIdentity` , call the `FilteredIdentity` helper method from the `QCAlgorithm` class. The `FilteredIdentity` method creates a `FilteredIdentity` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class FilteredIdentityAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.filteredidentity = self.FilteredIdentity(self.symbol, filter = lambda x: x.Close > x.Open)

    def OnData(self, slice: Slice) -> None:
        if self.filteredidentity.IsReady:
            # The current value of self.filteredidentity is represented by
self.filteredidentity.Current.Value
            self.Plot("FilteredIdentity", "filteredidentity", self.filteredidentity.Current.Value)
```

The following reference table describes the `FilteredIdentity` method:

Warning: include(/tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/qcalgorithm-api/filteredidentity.html): failed to open stream: No such file or directory in /tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/indicators/using-indicator.php on line 49
Warning: include(): Failed opening '/tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/qcalgorithm-api/filteredidentity.html' for inclusion (include_path='/var/www/beta/core/libraries/Google:/var/www/beta:.:/usr/share/php') in /tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/indicators/using-indicator.php on line 49

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `FilteredIdentity` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to

update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class FilteredIdentityAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.filteredidentity = FilteredIdentity("SPY", filter = lambda x: x.Close > x.Open)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.filteredidentity.Update(bar.EndTime, bar.Close)

        if self.filteredidentity.IsReady:
            # The current value of self.filteredidentity is represented by
self.filteredidentity.Current.Value
            self.Plot("FilteredIdentity", "filteredidentity", self.filteredidentity.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class FilteredIdentityAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.filteredidentity = FilteredIdentity("SPY", filter = lambda x: x.Close > x.Open)
        self.RegisterIndicator(self.symbol, self.filteredidentity, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.filteredidentity.IsReady:
            # The current value of self.filteredidentity is represented by
self.filteredidentity.Current.Value
            self.Plot("FilteredIdentity", "filteredidentity", self.filteredidentity.Current.Value)
```

The following reference table describes the `FilteredIdentity` constructor:

INDICATORS

FilteredIdentity() 1/1

```
FilteredIdentity QuantConnect.Indicators.FilteredIdentity (
string name,
Func filter
)
```

Initializes a new instance of the FilteredIdentity indicator with the specified name.

Show Details

Parameters		
`string`	name	The name of the indicator.
`Func`	filter	Filters the IBaseData send into the indicator, if null defaults to true (x =.

Return

`FilteredIdentity` - The new `FilteredIdentity` indicator object.

Definition at line 36 of file Indicators/FilteredIdentity.cs.

Visualization

The following image shows plot values of selected properties of `FilteredIdentity` using the plotly library.

Supported Indicators

Fisher Transform

Introduction

The Fisher transform is a mathematical process which is used to convert any data set to a modified data set whose Probability Distribution Function is approximately Gaussian. Once the Fisher transform is computed, the transformed data can then be analyzed in terms of it's deviation from the mean. The equation is y = .5 * ln [1 + x / 1 - x] where x is the input y is the output ln is the natural logarithm The Fisher transform has much sharper turning points than other indicators such as MACD For more info, read chapter 1 of Cybernetic Analysis for Stocks and Futures by John F. Ehlers We are implementing the latest version of this indicator found at Fig. 4 of http://www.mesasoftware.com/papers/UsingTheFisherTransform.pdf

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FISH Indicator

To create an automatic indicators for `FisherTransform` , call the `FISH` helper method from the `QCAlgorithm` class. The `FISH` method creates a `FisherTransform` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class FisherTransformAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.fish = self.FISH(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.fish.IsReady:
            # The current value of self.fish is represented by self.fish.Current.Value
            self.Plot("FisherTransform", "fish", self.fish.Current.Value)
```

The following reference table describes the `FISH` method:

INDICATORS

FISH() 1/1

```
FisherTransform QuantConnect.Algorithm.QCAlgorithm.FISH (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an FisherTransform indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose FisherTransform we want.
`Int32`	period	The period of the FisherTransform.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`FisherTransform` - The FisherTransform for the given parameters.

Definition at line 633 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `FisherTransform` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class FisherTransformAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.fish = FisherTransform(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.fish.Update(bar)

        if self.fish.IsReady:
            # The current value of self.fish is represented by self.fish.Current.Value
            self.Plot("FisherTransform", "fish", self.fish.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class FisherTransformAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.fish = FisherTransform(20)
        self.RegisterIndicator(self.symbol, self.fish, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.fish.IsReady:
            # The current value of self.fish is represented by self.fish.Current.Value
            self.Plot("FisherTransform", "fish", self.fish.Current.Value)
```

The following reference table describes the `FisherTransform` constructor:

INDICATORS

FisherTransform() 1/2

```
FisherTransform QuantConnect.Indicators.FisherTransform (
int period
)
```

Initializes a new instance of the FisherTransform class with the default name and period.

Show Details

Parameters		
int	period	The period of the WMA.

Return

`FisherTransform` - The new `FisherTransform` indicator object.

Definition at line 50 of file Indicators/FisherTransform.cs.

INDICATORS

FisherTransform() 2/2

```
FisherTransform QuantConnect.Indicators.FisherTransform (
string name,
int period
)
```

A Fisher Transform of Prices.

Show Details

Parameters		
string	name	string - the name of the indicator.
int	period	The number of periods for the indicator.

Return

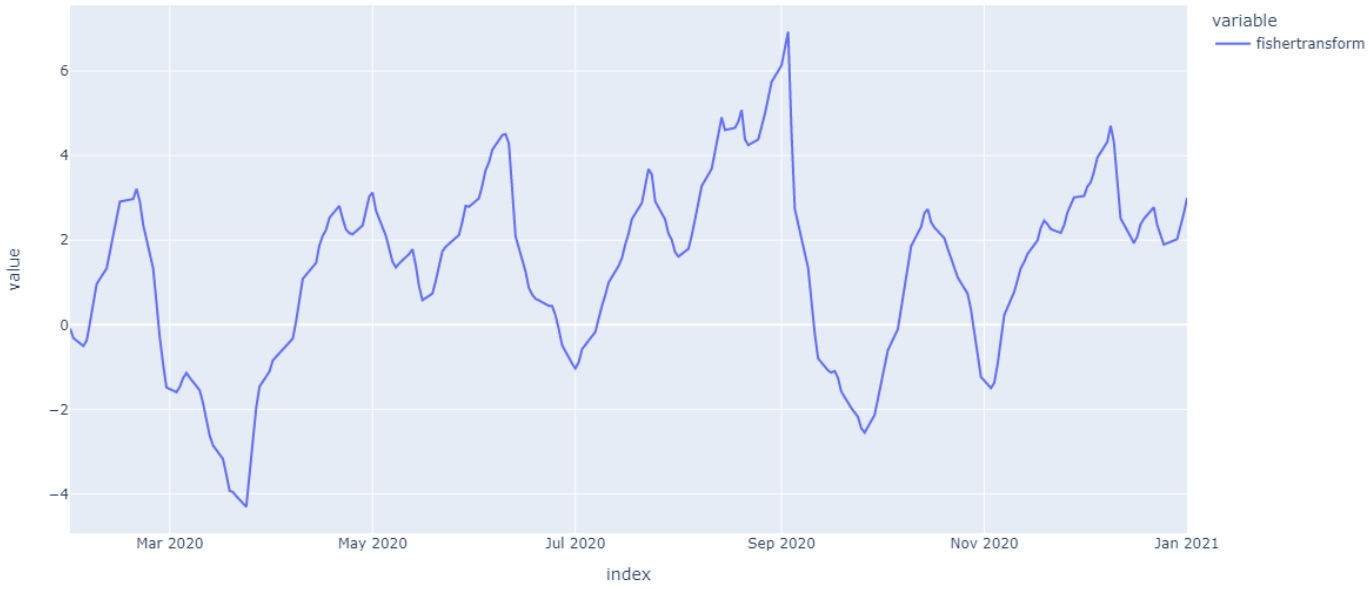
FisherTransform - The new FisherTransform indicator object.

Definition at line 60 of file Indicators/FisherTransform.cs.

Visualization

The following image shows plot values of selected properties of FisherTransform using the plotly library.

FISH("SPY", 20)


variable
fishertransform
value
6
4
2
0
−2
−4
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Fractal Adaptive Moving Average

Introduction

The Fractal Adaptive Moving Average (FRAMA) by John Ehlers

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FRAMA Indicator

To create an automatic indicators for `FractalAdaptiveMovingAverage` , call the `FRAMA` helper method from the `QCAlgorithm` class. The `FRAMA` method creates a `FractalAdaptiveMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class FractalAdaptiveMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.frama = self.FRAMA(self.symbol, 20, 198)

    def OnData(self, slice: Slice) -> None:
        if self.frama.IsReady:
            # The current value of self.frama is represented by self.frama.Current.Value
            self.Plot("FractalAdaptiveMovingAverage", "frama", self.frama.Current.Value)
```

The following reference table describes the `FRAMA` method:

INDICATORS

FRAMA() 1/1

```
FractalAdaptiveMovingAverage QuantConnect.Algorithm.QCAlgorithm.FRAMA (
Symbol symbol,
Int32 period,
*Int32 longPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates an FractalAdaptiveMovingAverage (FRAMA) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose FRAMA we want.
`Int32`	period	The period of the FRAMA.
`*Int32`	longPeriod	*(Optional)* The long period of the FRAMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`FractalAdaptiveMovingAverage` - The FRAMA for the given parameters.

Definition at line 653 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `FractalAdaptiveMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class FractalAdaptiveMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.frama = FractalAdaptiveMovingAverage(20, 198)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.frama.Update(bar)

        if self.frama.IsReady:
            # The current value of self.frama is represented by self.frama.Current.Value
            self.Plot("FractalAdaptiveMovingAverage", "frama", self.frama.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class FractalAdaptiveMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.frama = FractalAdaptiveMovingAverage(20, 198)
        self.RegisterIndicator(self.symbol, self.frama, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.frama.IsReady:
            # The current value of self.frama is represented by self.frama.Current.Value
            self.Plot("FractalAdaptiveMovingAverage", "frama", self.frama.Current.Value)
```

The following reference table describes the `FractalAdaptiveMovingAverage` constructor:

INDICATORS

FractalAdaptiveMovingAverage() 1/3

```
FractalAdaptiveMovingAverage QuantConnect.Indicators.FractalAdaptiveMovingAverage (
string name,
int n,
int longPeriod
)
```

Initializes a new instance of the average class.

Show Details

Parameters		
`string`	name	The name of the indicator instance.
`int`	n	The window period (must be even). Example value: 16.
`int`	longPeriod	The average period. Example value: 198.

Return

`FractalAdaptiveMovingAverage` - The new `FractalAdaptiveMovingAverage` indicator object.

Definition at line 38 of file Indicators/FractalAdaptiveMovingAverage.cs.

INDICATORS

FractalAdaptiveMovingAverage() 2/3

```
FractalAdaptiveMovingAverage QuantConnect.Indicators.FractalAdaptiveMovingAverage (
int n,
int longPeriod
)
```

Initializes a new instance of the average class.

Show Details

Parameters		
int	n	The window period (must be even). Example value: 16.
int	longPeriod	The average period. Example value: 198.

Return

`FractalAdaptiveMovingAverage` - The new `FractalAdaptiveMovingAverage` indicator object.

Definition at line 56 of file Indicators/FractalAdaptiveMovingAverage.cs.

INDICATORS

FractalAdaptiveMovingAverage() 3/3

```
FractalAdaptiveMovingAverage QuantConnect.Indicators.FractalAdaptiveMovingAverage (
int n
)
```

Initializes a new instance of the average class.

Show Details

Parameters		
int	n	The window period (must be even). Example value: 16.

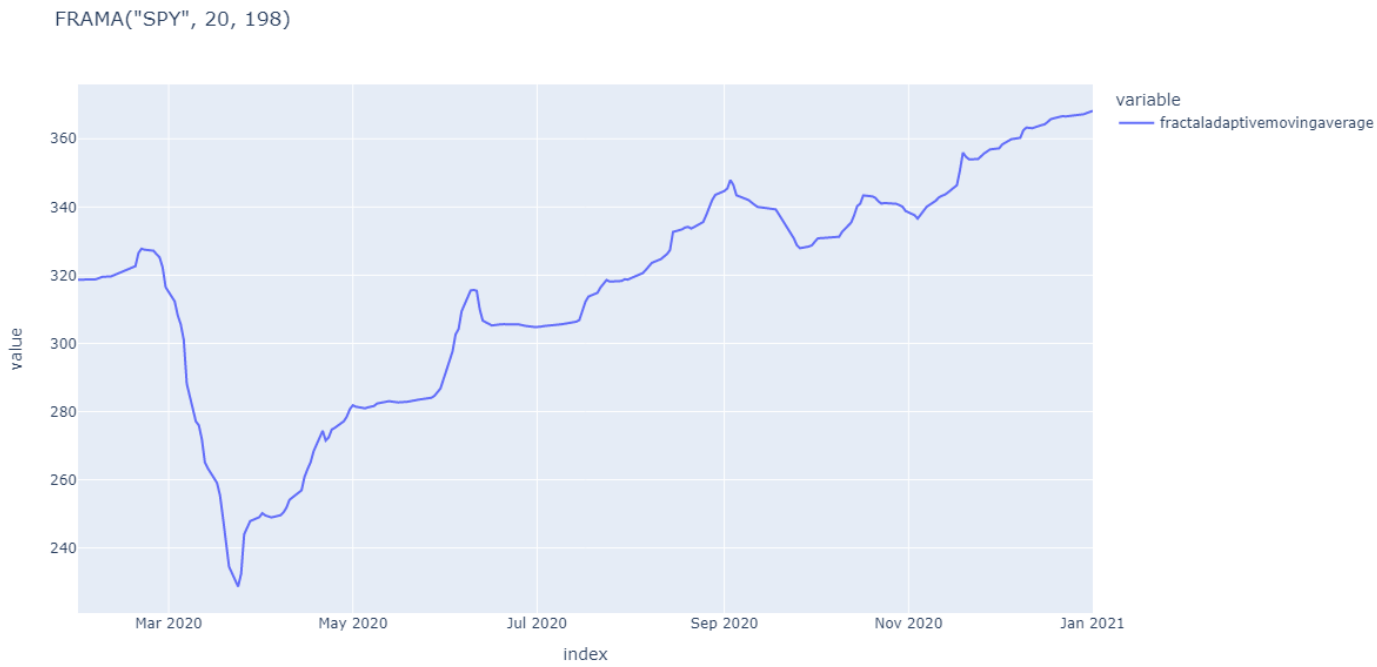
Return

`FractalAdaptiveMovingAverage` - The new `FractalAdaptiveMovingAverage` indicator object.

Definition at line 66 of file Indicators/FractalAdaptiveMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `FractalAdaptiveMovingAverage` using the plotly library.



Supported Indicators

Heikin Ashi

Introduction

This indicator computes the Heikin-Ashi bar (HA) The Heikin-Ashi bar is calculated using the following formulas: HA_Close[0] = (Open[0] + High[0] + Low[0] + Close[0]) / 4 HA_Open[0] = (HA_Open[1] + HA_Close[1]) / 2 HA_High[0] = MAX(High[0], HA_Open[0], HA_Close[0]) HA_Low[0] = MIN(Low[0], HA_Open[0], HA_Close[0])

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HeikinAshi Indicator

To create an automatic indicators for `HeikinAshi` , call the `HeikinAshi` helper method from the `QCAlgorithm` class. The `HeikinAshi` method creates a `HeikinAshi` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class HeikinAshiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.heikinashi = self.HeikinAshi(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.heikinashi.IsReady:
            # The current value of self.heikinashi is represented by self.heikinashi.Current.Value
            self.Plot("HeikinAshi", "heikinashi", self.heikinashi.Current.Value)
            # Plot all attributes of self.heikinashi
            self.Plot("HeikinAshi", "open", self.heikinashi.Open.Current.Value)
            self.Plot("HeikinAshi", "high", self.heikinashi.High.Current.Value)
            self.Plot("HeikinAshi", "low", self.heikinashi.Low.Current.Value)
            self.Plot("HeikinAshi", "close", self.heikinashi.Close.Current.Value)
            self.Plot("HeikinAshi", "volume", self.heikinashi.Volume.Current.Value)
```

The following reference table describes the `HeikinAshi` method:

Warning: include(/tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/qcalgorithm-api/heikinashi.html): failed to open stream: No such file or directory in /tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/indicators/using-indicator.php on line 49
Warning: include(): Failed opening '/tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/qcalgorithm-api/heikinashi.html' for inclusion (include_path='/var/www/beta/core/libraries/Google:/var/www/beta:.:/usr/share/php') in /tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/indicators/using-indicator.php on line 49

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HeikinAshi` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class HeikinAshiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.heikinashi = HeikinAshi()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.heikinashi.Update(bar)

        if self.heikinashi.IsReady:
            # The current value of self.heikinashi is represented by self.heikinashi.Current.Value
            self.Plot("HeikinAshi", "heikinashi", self.heikinashi.Current.Value)
            # Plot all attributes of self.heikinashi
            self.Plot("HeikinAshi", "open", self.heikinashi.Open.Current.Value)
            self.Plot("HeikinAshi", "high", self.heikinashi.High.Current.Value)
            self.Plot("HeikinAshi", "low", self.heikinashi.Low.Current.Value)
            self.Plot("HeikinAshi", "close", self.heikinashi.Close.Current.Value)
            self.Plot("HeikinAshi", "volume", self.heikinashi.Volume.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class HeikinAshiAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.heikinashi = HeikinAshi()
        self.RegisterIndicator(self.symbol, self.heikinashi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.heikinashi.IsReady:
            # The current value of self.heikinashi is represented by self.heikinashi.Current.Value
            self.Plot("HeikinAshi", "heikinashi", self.heikinashi.Current.Value)
            # Plot all attributes of self.heikinashi
            self.Plot("HeikinAshi", "open", self.heikinashi.Open.Current.Value)
            self.Plot("HeikinAshi", "high", self.heikinashi.High.Current.Value)
            self.Plot("HeikinAshi", "low", self.heikinashi.Low.Current.Value)
            self.Plot("HeikinAshi", "close", self.heikinashi.Close.Current.Value)
            self.Plot("HeikinAshi", "volume", self.heikinashi.Volume.Current.Value)
```

The following reference table describes the `HeikinAshi` constructor:

INDICATORS

HeikinAshi() 1/2

```
HeikinAshi QuantConnect.Indicators.HeikinAshi (
string name
)
```

Initializes a new instance of the `HeikinAsh` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`HeikinAshi` - The new `HeikinAshi` indicator object.

Definition at line 60 of file Indicators/HeikinAshi.cs.

INDICATORS

HeikinAshi() 2/2

```
HeikinAshi QuantConnect.Indicators.HeikinAshi (
)
```

Initializes a new instance of the `HeikinAsh` class.

Show Details

This method requires no argument input.

Return

`HeikinAshi` - The new `HeikinAshi` indicator object.

Definition at line 73 of file Indicators/HeikinAshi.cs.

Visualization

The following image shows plot values of selected properties of `HeikinAshi` using the plotly library.

HeikinAshi("SPY")

Supported Indicators

Hilbert Transform

Introduction

This indicator computes the Hilbert Transform Indicator by John Ehlers. By using present and prior price differences, and some feedback, price values are split into their complex number components of real (inPhase) and imaginary (quadrature) parts. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HilbertTransform Indicator

To create an automatic indicators for `HilbertTransform` , call the `HilbertTransform` helper method from the `QCAlgorithm` class. The `HilbertTransform` method creates a `HilbertTransform` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class HilbertTransformAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hilberttransform = self.HilbertTransform(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.hilberttransform.IsReady:
            # The current value of self.hilberttransform is represented by
self.hilberttransform.Current.Value
            self.Plot("HilbertTransform", "hilberttransform", self.hilberttransform.Current.Value)
            # Plot all attributes of self.hilberttransform
            self.Plot("HilbertTransform", "inphase", self.hilberttransform.InPhase.Current.Value)
            self.Plot("HilbertTransform", "quadrature", self.hilberttransform.Quadrature.Current.Value)
```

The following reference table describes the `HilbertTransform` method:

Warning: include(/tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/qcalgorithm-api/hilberttransform.html): failed to open stream: No such file or directory in /tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/indicators/using-indicator.php on line 49
Warning: include(): Failed opening '/tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/qcalgorithm-api/hilberttransform.html' for inclusion (include_path='/var/www/beta/core/libraries/Google:/var/www/beta:.:/usr/share/php') in /tmp/docs/227df3f244543bb12dcd97f1827791c4d3960ab7/Resources/indicators/using-indicator.php on line 49

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HilbertTransform` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```
PY
class HilbertTransformAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hilberttransform = HilbertTransform()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.hilberttransform.Update(bar.EndTime, bar.Close)

        if self.hilberttransform.IsReady:
            # The current value of self.hilberttransform is represented by
self.hilberttransform.Current.Value
            self.Plot("HilbertTransform", "hilberttransform", self.hilberttransform.Current.Value)
            # Plot all attributes of self.hilberttransform
            self.Plot("HilbertTransform", "inphase", self.hilberttransform.InPhase.Current.Value)
            self.Plot("HilbertTransform", "quadrature", self.hilberttransform.Quadrature.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
PY
class HilbertTransformAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hilberttransform = HilbertTransform()
        self.RegisterIndicator(self.symbol, self.hilberttransform, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.hilberttransform.IsReady:
            # The current value of self.hilberttransform is represented by
self.hilberttransform.Current.Value
            self.Plot("HilbertTransform", "hilberttransform", self.hilberttransform.Current.Value)
            # Plot all attributes of self.hilberttransform
            self.Plot("HilbertTransform", "inphase", self.hilberttransform.InPhase.Current.Value)
            self.Plot("HilbertTransform", "quadrature", self.hilberttransform.Quadrature.Current.Value)
```

The following reference table describes the `HilbertTransform` constructor:

INDICATORS

HilbertTransform() 1/2

```
HilbertTransform QuantConnect.Indicators.HilbertTransform (
string name
)
```

Creates a new Hilbert Transform indicator.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`HilbertTransform` - The new `HilbertTransform` indicator object.

Definition at line 67 of file Indicators/HilbertTransform.cs.

INDICATORS

HilbertTransform() 2/2

```
HilbertTransform QuantConnect.Indicators.HilbertTransform (
)
```

Creates a new Hilbert Transform indicator with default name and default params.

Show Details

This method requires no argument input.

Return

`HilbertTransform` - The new `HilbertTransform` indicator object.

Definition at line 137 of file Indicators/HilbertTransform.cs.

Visualization

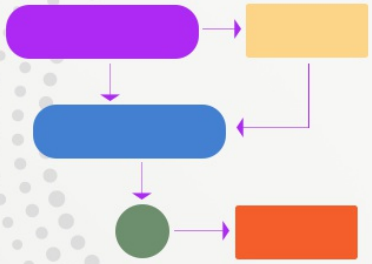
The following image shows plot values of selected properties of `HilbertTransform` using the plotly library.

WRITING ALGORITHMS

Indicators

Supported Indicators

Hilbert Transform



QUANTCONNECT

Supported Indicators

Hull Moving Average

Introduction

Produces a Hull Moving Average as explained at http://www.alanhull.com/hull-moving-average/ and derived from the instructions for the Excel VBA code at http://finance4traders.blogspot.com/2009/06/how-to-calculate-hull-moving-average.html

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HMA Indicator

To create an automatic indicators for `HullMovingAverage` , call the `HMA` helper method from the `QCAlgorithm` class. The `HMA` method creates a `HullMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class HullMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hma = self.HMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.hma.IsReady:
            # The current value of self.hma is represented by self.hma.Current.Value
            self.Plot("HullMovingAverage", "hma", self.hma.Current.Value)
```

The following reference table describes the `HMA` method:

INDICATORS

HMA() 1/1

```
HullMovingAverage QuantConnect.Algorithm.QCAlgorithm.HMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new HullMovingAverage indicator. The Hull moving average is a series of nested weighted moving averages, is fast and smooth.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Hull moving average we want.
`Int32`	period	The period over which to compute the Hull moving average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`HullMovingAverage` - The new `HullMovingAverage` object.

Definition at line 712 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `HullMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class HullMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hma = HullMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.hma.Update(bar.EndTime, bar.Close)

        if self.hma.IsReady:
            # The current value of self.hma is represented by self.hma.Current.Value
            self.Plot("HullMovingAverage", "hma", self.hma.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class HullMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.hma = HullMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.hma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.hma.IsReady:
            # The current value of self.hma is represented by self.hma.Current.Value
            self.Plot("HullMovingAverage", "hma", self.hma.Current.Value)
```

The following reference table describes the `HullMovingAverage` constructor:

INDICATORS

HullMovingAverage() 1/2

```
HullMovingAverage QuantConnect.Indicators.HullMovingAverage (
string name,
int period
)
```

A Hull Moving Average.

Show Details

Parameters		
`string`	name	string - a name for the indicator.
`int`	period	int - the number of periods to calculate the HMA - the period of the slower LWMA.

Return

`HullMovingAverage` - The new `HullMovingAverage` indicator object.

Definition at line 35 of file Indicators/HullMovingAverage.cs.

INDICATORS

HullMovingAverage() 2/2

```
HullMovingAverage QuantConnect.Indicators.HullMovingAverage (
int period
)
```

A Hull Moving Average.

Show Details

Parameters		
`int`	period	int - the number of periods over which to calculate the HMA - the length of the slower LWMA.

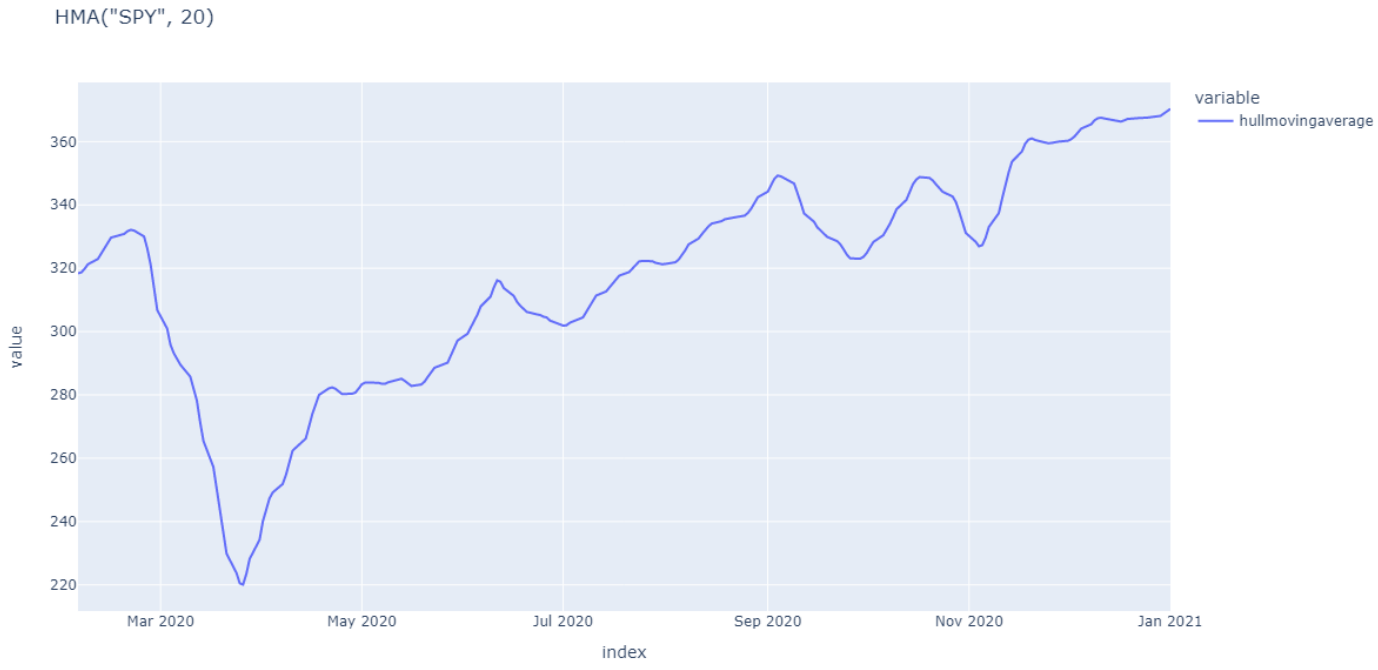
Return

`HullMovingAverage` - The new `HullMovingAverage` indicator object.

Definition at line 49 of file Indicators/HullMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `HullMovingAverage` using the plotly library.



Supported Indicators

Ichimoku Kinko Hyo

Introduction

This indicator computes the Ichimoku Kinko Hyo indicator. It consists of the following main indicators: Tenkan-sen: (Highest High + Lowest Low) / 2 for the specific period (normally 9) Kijun-sen: (Highest High + Lowest Low) / 2 for the specific period (normally 26) Senkou A Span: (Tenkan-sen + Kijun-sen)/ 2 from a specific number of periods ago (normally 26) Senkou B Span: (Highest High + Lowest Low) / 2 for the specific period (normally 52), from a specific number of periods ago (normally 26)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ICHIMOKU Indicator

To create an automatic indicators for `IchimokuKinkoHyo` , call the `ICHIMOKU` helper method from the `QCAlgorithm` class. The `ICHIMOKU` method creates a `IchimokuKinkoHyo` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class IchimokuKinkoHyoAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ichimoku = self.ICHIMOKU(self.symbol, 9, 26, 17, 52, 26, 26)

    def OnData(self, slice: Slice) -> None:
        if self.ichimoku.IsReady:
            # The current value of self.ichimoku is represented by self.ichimoku.Current.Value
            self.Plot("IchimokuKinkoHyo", "ichimoku", self.ichimoku.Current.Value)
            # Plot all attributes of self.ichimoku
            self.Plot("IchimokuKinkoHyo", "tenkan", self.ichimoku.Tenkan.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijun", self.ichimoku.Kijun.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoua", self.ichimoku.SenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoub", self.ichimoku.SenkouB.Current.Value)
            self.Plot("IchimokuKinkoHyo", "chikou", self.ichimoku.Chikou.Current.Value)
            self.Plot("IchimokuKinkoHyo", "tenkanmaximum", self.ichimoku.TenkanMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "tenkanminimum", self.ichimoku.TenkanMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijunmaximum", self.ichimoku.KijunMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijunminimum", self.ichimoku.KijunMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoubmaximum", self.ichimoku.SenkouBMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoubminimum", self.ichimoku.SenkouBMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedtenkansenkoua",
self.ichimoku.DelayedTenkanSenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedkijunsenkoua",
self.ichimoku.DelayedKijunSenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedmaximumsenkoub",
self.ichimoku.DelayedMaximumSenkouB.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedminimumsenkoub",
self.ichimoku.DelayedMinimumSenkouB.Current.Value)
```

The following reference table describes the `ICHIMOKU` method:

INDICATORS

ICHIMOKU() 1/1

```
IchimokuKinkoHyo QuantConnect.Algorithm.QCAlgorithm.ICHIMOKU (
Symbol symbol,
Int32 tenkanPeriod,
Int32 kijunPeriod,
Int32 senkouAPeriod,
Int32 senkouBPeriod,
Int32 senkouADelayPeriod,
Int32 senkouBDelayPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new IchimokuKinkoHyo indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose ICHIMOKU we want.
Int32	tenkanPeriod	The period to calculate the Tenkan-sen period.
Int32	kijunPeriod	The period to calculate the Kijun-sen period.
Int32	senkouAPeriod	The period to calculate the Tenkan-sen period.
Int32	senkouBPeriod	The period to calculate the Tenkan-sen period.
Int32	senkouADelayPeriod	The period to calculate the Tenkan-sen period.
Int32	senkouBDelayPeriod	The period to calculate the Tenkan-sen period.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, TradeBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

IchimokuKinkoHyo - A new IchimokuKinkoHyo indicator with the specified periods and delays.

Definition at line 736 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update

its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `IchimokuKinkoHyo` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```
class IchimokuKinkoHyoAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ichimoku = IchimokuKinkoHyo(9, 26, 17, 52, 26, 26)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ichimoku.Update(bar)

        if self.ichimoku.IsReady:
            # The current value of self.ichimoku is represented by self.ichimoku.Current.Value
            self.Plot("IchimokuKinkoHyo", "ichimoku", self.ichimoku.Current.Value)
            # Plot all attributes of self.ichimoku
            self.Plot("IchimokuKinkoHyo", "tenkan", self.ichimoku.Tenkan.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijun", self.ichimoku.Kijun.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoua", self.ichimoku.SenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoub", self.ichimoku.SenkouB.Current.Value)
            self.Plot("IchimokuKinkoHyo", "chikou", self.ichimoku.Chikou.Current.Value)
            self.Plot("IchimokuKinkoHyo", "tenkanmaximum", self.ichimoku.TenkanMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "tenkanminimum", self.ichimoku.TenkanMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijunmaximum", self.ichimoku.KijunMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijunminimum", self.ichimoku.KijunMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoubmaximum", self.ichimoku.SenkouBMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoubminimum", self.ichimoku.SenkouBMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedtenkansenkoua",
self.ichimoku.DelayedTenkanSenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedkijunsenkoua",
self.ichimoku.DelayedKijunSenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedmaximumsenkoub",
self.ichimoku.DelayedMaximumSenkouB.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedminimumsenkoub",
self.ichimoku.DelayedMinimumSenkouB.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class IchimokuKinkoHyoAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ichimoku = IchimokuKinkoHyo(9, 26, 17, 52, 26, 26)
        self.RegisterIndicator(self.symbol, self.ichimoku, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ichimoku.IsReady:
            # The current value of self.ichimoku is represented by self.ichimoku.Current.Value
            self.Plot("IchimokuKinkoHyo", "ichimoku", self.ichimoku.Current.Value)
            # Plot all attributes of self.ichimoku
            self.Plot("IchimokuKinkoHyo", "tenkan", self.ichimoku.Tenkan.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijun", self.ichimoku.Kijun.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoua", self.ichimoku.SenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoub", self.ichimoku.SenkouB.Current.Value)
            self.Plot("IchimokuKinkoHyo", "chikou", self.ichimoku.Chikou.Current.Value)
            self.Plot("IchimokuKinkoHyo", "tenkanmaximum", self.ichimoku.TenkanMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "tenkanminimum", self.ichimoku.TenkanMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijunmaximum", self.ichimoku.KijunMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "kijunminimum", self.ichimoku.KijunMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoubmaximum", self.ichimoku.SenkouBMaximum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "senkoubminimum", self.ichimoku.SenkouBMinimum.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedtenkansenkoua",
self.ichimoku.DelayedTenkanSenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedkijunsenkoua",
self.ichimoku.DelayedKijunSenkouA.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedmaximumsenkoub",
self.ichimoku.DelayedMaximumSenkouB.Current.Value)
            self.Plot("IchimokuKinkoHyo", "delayedminimumsenkoub",
self.ichimoku.DelayedMinimumSenkouB.Current.Value)
```

The following reference table describes the `IchimokuKinkoHyo` constructor:

INDICATORS

IchimokuKinkoHyo() 1/2

```
IchimokuKinkoHyo QuantConnect.Indicators.IchimokuKinkoHyo (
*int tenkanPeriod,
*int kijunPeriod,
*int senkouAPeriod,
*int senkouBPeriod,
*int senkouADelayPeriod,
*int senkouBDelayPeriod
)
```

Creates a new IchimokuKinkoHyo indicator from the specific periods.

Show Details

Parameters		
*int	tenkanPeriod	*(Optional) (Optional)* The Tenkan-sen period. Default: 9.
*int	kijunPeriod	*(Optional) (Optional)* The Kijun-sen period. Default: 26.
*int	senkouAPeriod	*(Optional) (Optional)* The Senkou A Span period. Default: 26.
*int	senkouBPeriod	*(Optional) (Optional)* The Senkou B Span period. Default: 52.
*int	senkouADelayPeriod	*(Optional) (Optional)* The Senkou A Span delay. Default: 26.
*int	senkouBDelayPeriod	*(Optional) (Optional)* The Senkou B Span delay. Default: 26.

Return

`IchimokuKinkoHyo` - The new `IchimokuKinkoHyo` indicator object.

Definition at line 114 of file Indicators/IchimokuKinkoHyo.cs.

INDICATORS

IchimokuKinkoHyo() 2/2

```
IchimokuKinkoHyo QuantConnect.Indicators.IchimokuKinkoHyo (
 string name,
 *int tenkanPeriod,
 *int kijunPeriod,
 *int senkouAPeriod,
 *int senkouBPeriod,
 *int senkouADelayPeriod,
 *int senkouBDelayPeriod
)
```

Creates a new IchimokuKinkoHyo indicator from the specific periods.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*int`	tenkanPeriod	*(Optional) (Optional)* The Tenkan-sen period. Default: 9.
`*int`	kijunPeriod	*(Optional) (Optional)* The Kijun-sen period. Default: 26.
`*int`	senkouAPeriod	*(Optional) (Optional)* The Senkou A Span period. Default: 26.
`*int`	senkouBPeriod	*(Optional) (Optional)* The Senkou B Span period. Default: 52.
`*int`	senkouADelayPeriod	*(Optional) (Optional)* The Senkou A Span delay. Default: 26.
`*int`	senkouBDelayPeriod	*(Optional) (Optional)* The Senkou B Span delay. Default: 26.

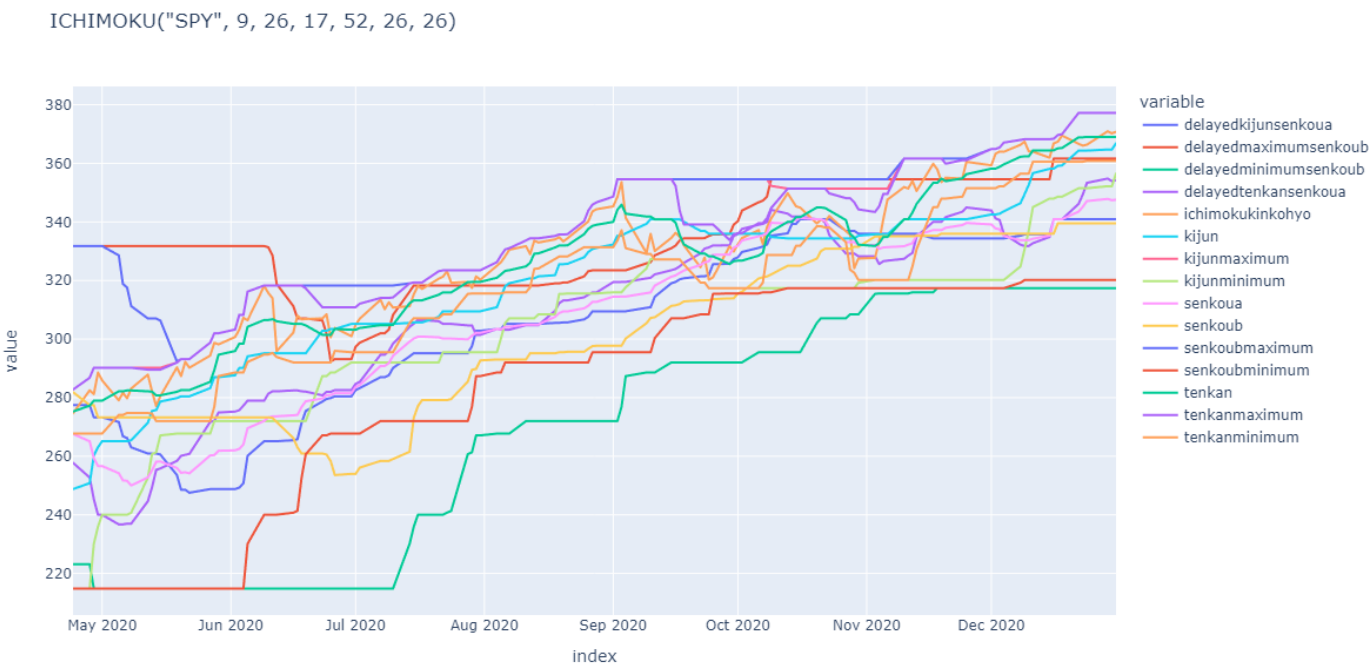
Return

`IchimokuKinkoHyo` - The new `IchimokuKinkoHyo` indicator object.

Definition at line 138 of file Indicators/IchimokuKinkoHyo.cs.

Visualization

The following image shows plot values of selected properties of `IchimokuKinkoHyo` using the plotly library.



Supported Indicators

Identity

Introduction

This indicator represents an indicator that is a ready after ingesting a single sample and always returns the same value as it is given.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using Identity Indicator

To create an automatic indicators for `Identity` , call the `Identity` helper method from the `QCAlgorithm` class. The `Identity` method creates a `Identity` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class IdentityAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.identity = self.Identity(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.identity.IsReady:
            # The current value of self.identity is represented by self.identity.Current.Value
            self.Plot("Identity", "identity", self.identity.Current.Value)
```

The following reference table describes the `Identity` method:

INDICATORS

Identity() 1/3

```
Identity QuantConnect.Algorithm.QCAlgorithm.Identity (
Symbol symbol,
*Func<IBaseData, Decimal> selector,
*String fieldName
)
```

Creates a new Identity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`Identity` - A new Identity indicator for the specified symbol and selector.

Definition at line 755 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

Identity() 2/3

```
Identity QuantConnect.Algorithm.QCAlgorithm.Identity (
Symbol symbol,
Resolution resolution,
*Func<IBaseData, Decimal> selector,
*String fieldName
)
```

Creates a new Identity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`Resolution`	resolution	The desired resolution of the data.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`Identity` - A new Identity indicator for the specified symbol and selector.

Definition at line 771 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

Identity() 3/3

```
Identity QuantConnect.Algorithm.QCAlgorithm.Identity (
Symbol symbol,
TimeSpan resolution,
*Func<IBaseData, Decimal> selector,
*String fieldName
)
```

Creates a new Identity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`TimeSpan`	resolution	The desired resolution of the data.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`Identity` - A new Identity indicator for the specified symbol and selector.

Definition at line 789 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Identity` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class IdentityAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.identity = Identity("SPY")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.identity.Update(bar.EndTime, bar.Close)

        if self.identity.IsReady:
            # The current value of self.identity is represented by self.identity.Current.Value
            self.Plot("Identity", "identity", self.identity.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class IdentityAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.identity = Identity("SPY")
        self.RegisterIndicator(self.symbol, self.identity, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.identity.IsReady:
            # The current value of self.identity is represented by self.identity.Current.Value
            self.Plot("Identity", "identity", self.identity.Current.Value)
```

The following reference table describes the `Identity` constructor:

INDICATORS

Identity() 1/1

```
Identity QuantConnect.Indicators.Identity (
string name
)
```

Initializes a new instance of the Identity indicator with the specified name.

Show Details

Parameters		
`string`	name	The name of the indicator.

Return

`Identity` - The new `Identity` indicator object.

Definition at line 28 of file Indicators/Identity.cs.

Visualization

The following image shows plot values of selected properties of `Identity` using the plotly library.

Supported Indicators

Intraday Vwap

Introduction

Defines the canonical intraday VWAP indicator

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VWAP Indicator

To create an automatic indicators for `IntradayVwap` , call the `VWAP` helper method from the `QCAlgorithm` class. The `VWAP` method creates a `IntradayVwap` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class IntradayVwapAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vwap = self.VWAP(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.vwap.IsReady:
            # The current value of self.vwap is represented by self.vwap.Current.Value
            self.Plot("IntradayVwap", "vwap", self.vwap.Current.Value)
```

The following reference table describes the `VWAP` method:

INDICATORS

VWAP() 1/2

```
VolumeWeightedAveragePriceIndicator QuantConnect.Algorithm.QCAlgorithm.VWAP (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an VolumeWeightedAveragePrice (VWAP) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VWAP we want.
`Int32`	period	The period of the VWAP.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`VolumeWeightedAveragePriceIndicator` - The VolumeWeightedAveragePrice for the given parameters.

Definition at line 1831 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

VWAP() 2/2

```
IntradayVwap QuantConnect.Algorithm.QCAlgorithm.VWAP (
 Symbol symbol
)
```

Creates the canonical VWAP indicator that resets each day. The indicator will be automatically updated on the security's configured resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VWAP we want.

Return

`IntradayVwap` - The IntradayVWAP for the specified symbol.

Definition at line 1847 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update

its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `IntradayVwap` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```
PY
class IntradayVwapAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vwap = IntradayVwap("SPY")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.vwap.Update(bar.EndTime, bar.Close)

        if self.vwap.IsReady:
            # The current value of self.vwap is represented by self.vwap.Current.Value
            self.Plot("IntradayVwap", "vwap", self.vwap.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
PY
class IntradayVwapAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vwap = IntradayVwap("SPY")
        self.RegisterIndicator(self.symbol, self.vwap, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.vwap.IsReady:
            # The current value of self.vwap is represented by self.vwap.Current.Value
            self.Plot("IntradayVwap", "vwap", self.vwap.Current.Value)
```

The following reference table describes the `IntradayVwap` constructor:

INDICATORS

IntradayVwap() 1/1

```
IntradayVwap QuantConnect.Indicators.IntradayVwap (
string name
)
```

Initializes a new instance of the `IntradayVwa` class.

Show Details

Parameters		
`string`	name	The name of the indicator.

Return

`IntradayVwap` - The new `IntradayVwap` indicator object.

Definition at line 25 of file Indicators/IntradayVwap.cs.

Visualization

The following image shows plot values of selected properties of `IntradayVwap` using the plotly library.

Supported Indicators

Kaufman Adaptive Moving Average

Introduction

This indicator computes the Kaufman Adaptive Moving Average (KAMA). The Kaufman Adaptive Moving Average is calculated as explained here: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:kaufman_s_adaptive_moving_average

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KAMA Indicator

To create an automatic indicators for `KaufmanAdaptiveMovingAverage` , call the `KAMA` helper method from the `QCAlgorithm` class. The `KAMA` method creates a `KaufmanAdaptiveMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class KaufmanAdaptiveMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kama = self.KAMA(self.symbol, 20, 10, 20)

    def OnData(self, slice: Slice) -> None:
        if self.kama.IsReady:
            # The current value of self.kama is represented by self.kama.Current.Value
            self.Plot("KaufmanAdaptiveMovingAverage", "kama", self.kama.Current.Value)
```

The following reference table describes the `KAMA` method:

INDICATORS

KAMA() 1/2

```
KaufmanAdaptiveMovingAverage QuantConnect.Algorithm.QCAlgorithm.KAMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new KaufmanAdaptiveMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose KAMA we want.
`Int32`	period	The period of the Efficiency Ratio (ER) of KAMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`KaufmanAdaptiveMovingAverage` - The KaufmanAdaptiveMovingAverage indicator for the requested symbol over the specified period.

Definition at line 806 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

KAMA() 2/2

```
KaufmanAdaptiveMovingAverage QuantConnect.Algorithm.QCAlgorithm.KAMA (
 Symbol symbol,
 Int32 period,
 Int32 fastEmaPeriod,
 Int32 slowEmaPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new KaufmanAdaptiveMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose KAMA we want.
`Int32`	period	The period of the Efficiency Ratio (ER).
`Int32`	fastEmaPeriod	The period of the fast EMA used to calculate the Smoothing Constant (SC).
`Int32`	slowEmaPeriod	The period of the slow EMA used to calculate the Smoothing Constant (SC).
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`KaufmanAdaptiveMovingAverage` - The KaufmanAdaptiveMovingAverage indicator for the requested symbol over the specified period.

Definition at line 822 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `KaufmanAdaptiveMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class KaufmanAdaptiveMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kama = KaufmanAdaptiveMovingAverage(20, 10, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.kama.Update(bar.EndTime, bar.Close)

        if self.kama.IsReady:
            # The current value of self.kama is represented by self.kama.Current.Value
            self.Plot("KaufmanAdaptiveMovingAverage", "kama", self.kama.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class KaufmanAdaptiveMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kama = KaufmanAdaptiveMovingAverage(20, 10, 20)
        self.RegisterIndicator(self.symbol, self.kama, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.kama.IsReady:
            # The current value of self.kama is represented by self.kama.Current.Value
            self.Plot("KaufmanAdaptiveMovingAverage", "kama", self.kama.Current.Value)
```

The following reference table describes the `KaufmanAdaptiveMovingAverage` constructor:

INDICATORS

KaufmanAdaptiveMovingAverage() 1/2

```
KaufmanAdaptiveMovingAverage QuantConnect.Indicators.KaufmanAdaptiveMovingAverage (
string name,
int period,
*int fastEmaPeriod,
*int slowEmaPeriod
)
```

Initializes a new instance of the `KaufmanAdaptiveMovingAverag` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the Efficiency Ratio (ER).
`*int`	fastEmaPeriod	*(Optional) (Optional)* The period of the fast EMA used to calculate the Smoothing Constant (SC). Default: 2.
`*int`	slowEmaPeriod	*(Optional) (Optional)* The period of the slow EMA used to calculate the Smoothing Constant (SC). Default: 30.

Return

`KaufmanAdaptiveMovingAverage` - The new `KaufmanAdaptiveMovingAverage` indicator object.

Definition at line 36 of file Indicators/KaufmanAdaptiveMovingAverage.cs.

INDICATORS

KaufmanAdaptiveMovingAverage() 2/2

```
KaufmanAdaptiveMovingAverage QuantConnect.Indicators.KaufmanAdaptiveMovingAverage (
 int period,
 *int fastEmaPeriod,
 *int slowEmaPeriod
)
```

Initializes a new instance of the `KaufmanAdaptiveMovingAverag` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the Efficiency Ratio (ER).
`*int`	fastEmaPeriod	*(Optional) (Optional)* The period of the fast EMA used to calculate the Smoothing Constant (SC). Default: 2.
`*int`	slowEmaPeriod	*(Optional) (Optional)* The period of the slow EMA used to calculate the Smoothing Constant (SC). Default: 30.

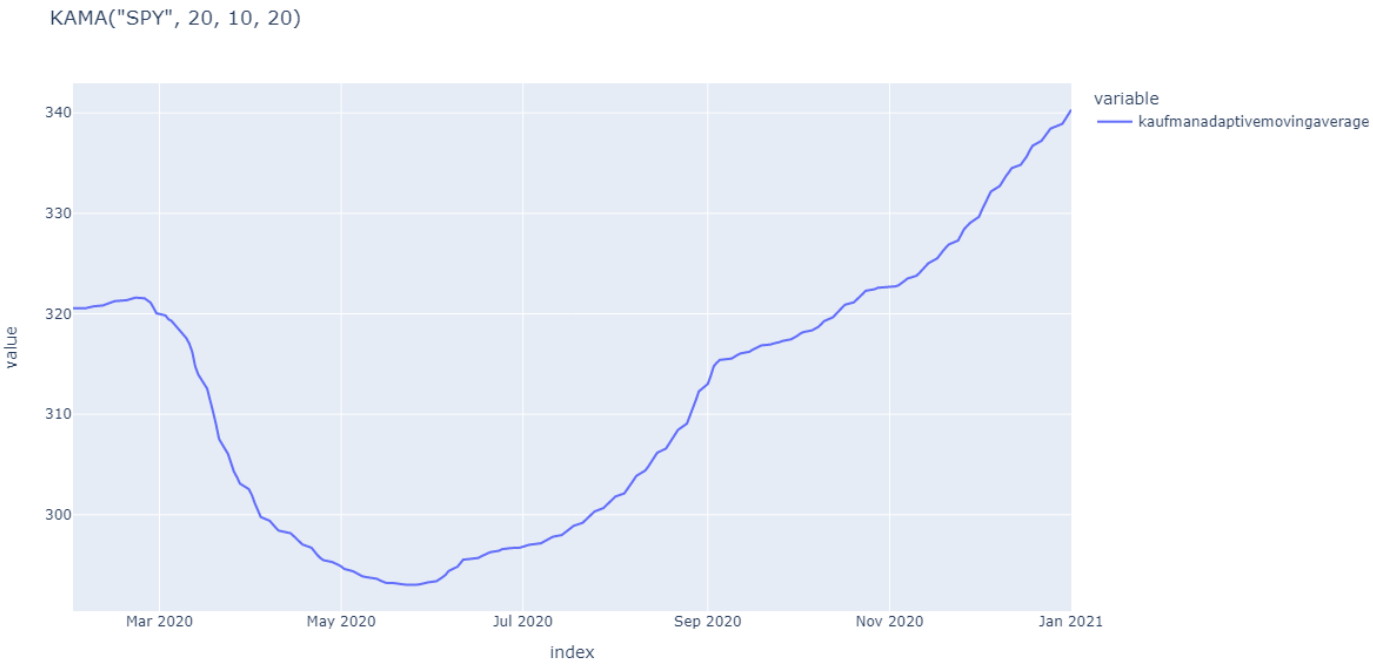
Return

`KaufmanAdaptiveMovingAverage` - The new `KaufmanAdaptiveMovingAverage` indicator object.

Definition at line 51 of file Indicators/KaufmanAdaptiveMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `KaufmanAdaptiveMovingAverage` using the plotly library.



Supported Indicators

Kaufman Efficiency Ratio

Introduction

This indicator computes the Kaufman Efficiency Ratio (KER). The Kaufman Efficiency Ratio is calculated as explained here: https://www.marketvolume.com/technicalanalysis/efficiencyratio.asp

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KER Indicator

To create an automatic indicators for `KaufmanEfficiencyRatio` , call the `KER` helper method from the `QCAlgorithm` class. The `KER` method creates a `KaufmanEfficiencyRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class KaufmanEfficiencyRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ker = self.KER(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.ker.IsReady:
            # The current value of self.ker is represented by self.ker.Current.Value
            self.Plot("KaufmanEfficiencyRatio", "ker", self.ker.Current.Value)
```

The following reference table describes the `KER` method:

INDICATORS

KER() 1/1

```
KaufmanEfficiencyRatio QuantConnect.Algorithm.QCAlgorithm.KER (
Symbol symbol,
*Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an KaufmanEfficiencyRatio indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EF we want.
`*Int32`	period	*(Optional)* The period of the EF.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`KaufmanEfficiencyRatio` - The KaufmanEfficiencyRatio indicator for the given parameters.

Definition at line 841 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `KaufmanEfficiencyRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class KaufmanEfficiencyRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ker = KaufmanEfficiencyRatio(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ker.Update(bar.EndTime, bar.Close)

        if self.ker.IsReady:
            # The current value of self.ker is represented by self.ker.Current.Value
            self.Plot("KaufmanEfficiencyRatio", "ker", self.ker.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class KaufmanEfficiencyRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ker = KaufmanEfficiencyRatio(20)
        self.RegisterIndicator(self.symbol, self.ker, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ker.IsReady:
            # The current value of self.ker is represented by self.ker.Current.Value
            self.Plot("KaufmanEfficiencyRatio", "ker", self.ker.Current.Value)
```

The following reference table describes the `KaufmanEfficiencyRatio` constructor:

INDICATORS

KaufmanEfficiencyRatio() 1/2

```
KaufmanEfficiencyRatio QuantConnect.Indicators.KaufmanEfficiencyRatio (
string name,
int period
)
```

Initializes a new instance of the `KaufmanEfficiencyRati` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the Efficiency Ratio (ER).

Return

`KaufmanEfficiencyRatio` - The new `KaufmanEfficiencyRatio` indicator object.

Definition at line 41 of file Indicators/KaufmanEfficiencyRatio.cs.

INDICATORS

KaufmanEfficiencyRatio() 2/2

```
KaufmanEfficiencyRatio QuantConnect.Indicators.KaufmanEfficiencyRatio (
int period
)
```

Initializes a new instance of the `KaufmanEfficiencyRati` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the Efficiency Ratio (ER).

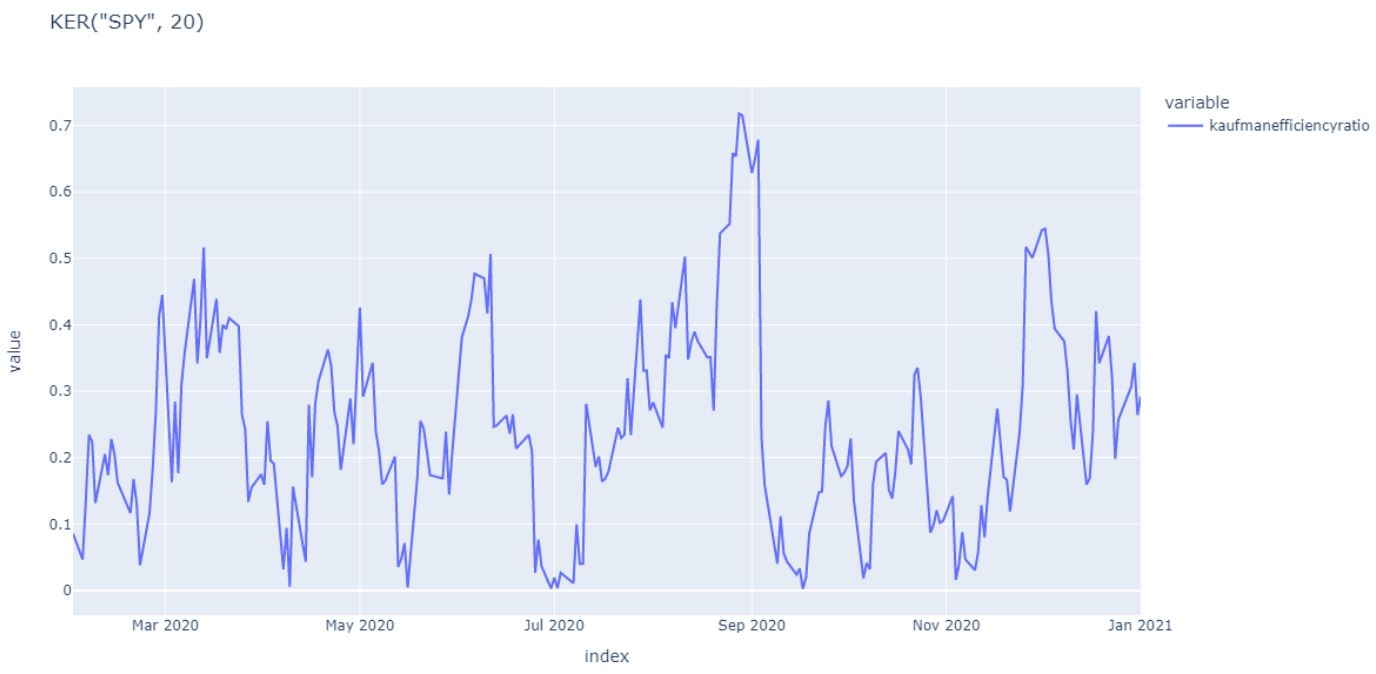
Return

`KaufmanEfficiencyRatio` - The new `KaufmanEfficiencyRatio` indicator object.

Definition at line 50 of file Indicators/KaufmanEfficiencyRatio.cs.

Visualization

The following image shows plot values of selected properties of `KaufmanEfficiencyRatio` using the plotly library.



Supported Indicators

Keltner Channels

Introduction

This indicator creates a moving average (middle band) with an upper band and lower band fixed at k average True range multiples away from the middle band.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KCH Indicator

To create an automatic indicators for `KeltnerChannels` , call the `KCH` helper method from the `QCAlgorithm` class. The `KCH` method creates a `KeltnerChannels` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class KeltnerChannelsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kch = self.KCH(self.symbol, 20, 2, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.kch.IsReady:
            # The current value of self.kch is represented by self.kch.Current.Value
            self.Plot("KeltnerChannels", "kch", self.kch.Current.Value)
            # Plot all attributes of self.kch
            self.Plot("KeltnerChannels", "middleband", self.kch.MiddleBand.Current.Value)
            self.Plot("KeltnerChannels", "upperband", self.kch.UpperBand.Current.Value)
            self.Plot("KeltnerChannels", "lowerband", self.kch.LowerBand.Current.Value)
            self.Plot("KeltnerChannels", "averagetruerange", self.kch.AverageTrueRange.Current.Value)
```

The following reference table describes the `KCH` method:

INDICATORS

KCH() 1/1

```
KeltnerChannels QuantConnect.Algorithm.QCAlgorithm.KCH (
Symbol symbol,
Int32 period,
Decimal k,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Keltner Channels indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Keltner Channel we seek.
`Int32`	period	The period over which to compute the Keltner Channels.
`Decimal`	k	The number of multiples of the `AverageTrueRange` from the middle band of the Keltner Channels.
`*MovingAverageType`	movingAverageType	*(Optional)* Specifies the type of moving average to be used as the middle line of the Keltner Channel.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`KeltnerChannels` - The Keltner Channel indicator for the requested symbol.

Definition at line 862 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `KeltnerChannels` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class KeltnerChannelsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kch = KeltnerChannels(20, 2, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.kch.Update(bar)

        if self.kch.IsReady:
            # The current value of self.kch is represented by self.kch.Current.Value
            self.Plot("KeltnerChannels", "kch", self.kch.Current.Value)
            # Plot all attributes of self.kch
            self.Plot("KeltnerChannels", "middleband", self.kch.MiddleBand.Current.Value)
            self.Plot("KeltnerChannels", "upperband", self.kch.UpperBand.Current.Value)
            self.Plot("KeltnerChannels", "lowerband", self.kch.LowerBand.Current.Value)
            self.Plot("KeltnerChannels", "averagetruerange", self.kch.AverageTrueRange.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class KeltnerChannelsAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.kch = KeltnerChannels(20, 2, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.kch, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.kch.IsReady:
            # The current value of self.kch is represented by self.kch.Current.Value
            self.Plot("KeltnerChannels", "kch", self.kch.Current.Value)
            # Plot all attributes of self.kch
            self.Plot("KeltnerChannels", "middleband", self.kch.MiddleBand.Current.Value)
            self.Plot("KeltnerChannels", "upperband", self.kch.UpperBand.Current.Value)
            self.Plot("KeltnerChannels", "lowerband", self.kch.LowerBand.Current.Value)
            self.Plot("KeltnerChannels", "averagetruerange", self.kch.AverageTrueRange.Current.Value)
```

The following reference table describes the `KeltnerChannels` constructor:

INDICATORS

KeltnerChannels() 1/2

```
KeltnerChannels QuantConnect.Indicators.KeltnerChannels (
int period,
decimal k,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the KeltnerChannels class.

Show Details

Parameters		
int	period	The period of the average true range and moving average (middle band).
decimal	k	The number of multiplies specifying the distance between the middle band and upper or lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used. Default: MovingAverageType.Simple.

Return

`KeltnerChannels` - The new `KeltnerChannels` indicator object.

Definition at line 53 of file Indicators/KeltnerChannels.cs.

INDICATORS

KeltnerChannels() 2/2

```
KeltnerChannels QuantConnect.Indicators.KeltnerChannels (
string name,
int period,
decimal k,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the KeltnerChannels class.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the average true range and moving average (middle band).
decimal	k	The number of multiples specifying the distance between the middle band and upper or lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used. Default: MovingAverageType.Simple.

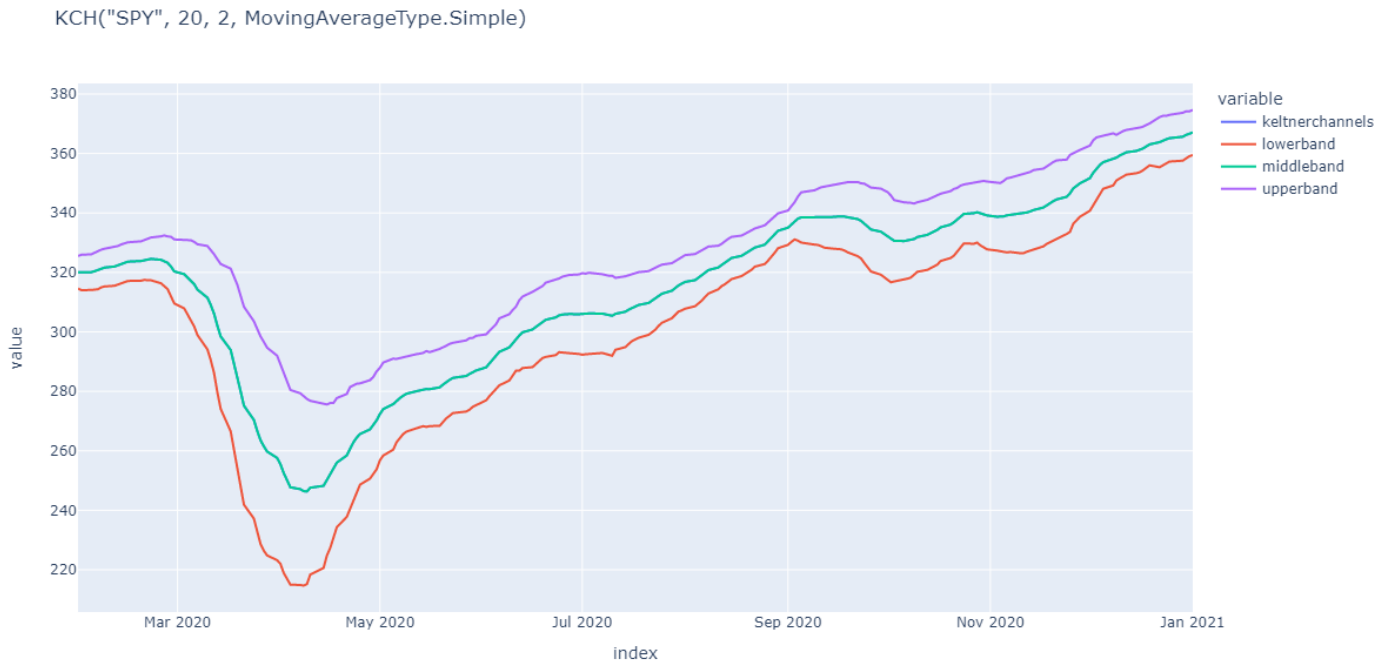
Return

`KeltnerChannels` - The new `KeltnerChannels` indicator object.

Definition at line 65 of file Indicators/KeltnerChannels.cs.

Visualization

The following image shows plot values of selected properties of `KeltnerChannels` using the plotly library.



Supported Indicators

Least Squares Moving Average

Introduction

The Least Squares Moving Average (LSMA) first calculates a least squares regression line over the preceding time periods, and then projects it forward to the current period. In essence, it calculates what the value would be if the regression line continued. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using LSMA Indicator

To create an automatic indicators for `LeastSquaresMovingAverage` , call the `LSMA` helper method from the `QCAlgorithm` class. The `LSMA` method creates a `LeastSquaresMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class LeastSquaresMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.lsma = self.LSMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.lsma.IsReady:
            # The current value of self.lsma is represented by self.lsma.Current.Value
            self.Plot("LeastSquaresMovingAverage", "lsma", self.lsma.Current.Value)
            # Plot all attributes of self.lsma
            self.Plot("LeastSquaresMovingAverage", "intercept", self.lsma.Intercept.Current.Value)
            self.Plot("LeastSquaresMovingAverage", "slope", self.lsma.Slope.Current.Value)
```

The following reference table describes the `LSMA` method:

INDICATORS

LSMA() 1/1

```
LeastSquaresMovingAverage QuantConnect.Algorithm.QCAlgorithm.LSMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates and registers a new Least Squares Moving Average instance.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose LSMA we seek.
`Int32`	period	The LSMA period. Normally 14.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`LeastSquaresMovingAverage` - A LeastSquaredMovingAverage configured with the specified period.

Definition at line 898 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `LeastSquaresMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class LeastSquaresMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.lsma = LeastSquaresMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.lsma.Update(bar.EndTime, bar.Close)

        if self.lsma.IsReady:
            # The current value of self.lsma is represented by self.lsma.Current.Value
            self.Plot("LeastSquaresMovingAverage", "lsma", self.lsma.Current.Value)
            # Plot all attributes of self.lsma
            self.Plot("LeastSquaresMovingAverage", "intercept", self.lsma.Intercept.Current.Value)
            self.Plot("LeastSquaresMovingAverage", "slope", self.lsma.Slope.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class LeastSquaresMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.lsma = LeastSquaresMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.lsma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.lsma.IsReady:
            # The current value of self.lsma is represented by self.lsma.Current.Value
            self.Plot("LeastSquaresMovingAverage", "lsma", self.lsma.Current.Value)
            # Plot all attributes of self.lsma
            self.Plot("LeastSquaresMovingAverage", "intercept", self.lsma.Intercept.Current.Value)
            self.Plot("LeastSquaresMovingAverage", "slope", self.lsma.Slope.Current.Value)
```

The following reference table describes the LeastSquaresMovingAverage constructor:

INDICATORS

LeastSquaresMovingAverage() 1/2

```
LeastSquaresMovingAverage QuantConnect.Indicators.LeastSquaresMovingAverage (
string name,
int period
)
```

Initializes a new instance of the LeastSquaresMovingAverag class.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The number of data points to hold in the window.

Return

LeastSquaresMovingAverage - The new LeastSquaresMovingAverage indicator object.

Definition at line 56 of file Indicators/LeastSquaresMovingAverage.cs.

INDICATORS

LeastSquaresMovingAverage() 2/2

```
LeastSquaresMovingAverage QuantConnect.Indicators.LeastSquaresMovingAverage (
int period
)
```

Initializes a new instance of the `LeastSquaresMovingAverag` class.

Show Details

Parameters		
`int`	period	The number of data points to hold in the window.

Return

`LeastSquaresMovingAverage` - The new `LeastSquaresMovingAverage` indicator object.

Definition at line 68 of file Indicators/LeastSquaresMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `LeastSquaresMovingAverage` using the plotly library.

Supported Indicators

Linear Weighted Moving Average

Introduction

This indicator represents the traditional Weighted Moving Average indicator. The weight are linearly distributed according to the number of periods in the indicator. For example, a 4 period indicator will have a numerator of (4 * window[0]) + (3 * window[1]) + (2 * window[2]) + window[3] and a denominator of 4 + 3 + 2 + 1 = 10 During the warm up period, IsReady will return False, but the LWMA will still be computed correctly because the denominator will be the minimum of Samples factorial or Size factorial and the computation iterates over that minimum value. The RollingWindow of inputs is created when the indicator is created. A RollingWindow of LWMAs is not saved. That is up to the caller.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using LWMA Indicator

To create an automatic indicators for `LinearWeightedMovingAverage` , call the `LWMA` helper method from the `QCAlgorithm` class. The `LWMA` method creates a `LinearWeightedMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class LinearWeightedMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.lwma = self.LWMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.lwma.IsReady:
            # The current value of self.lwma is represented by self.lwma.Current.Value
            self.Plot("LinearWeightedMovingAverage", "lwma", self.lwma.Current.Value)
```

The following reference table describes the `LWMA` method:

INDICATORS

LWMA() 1/1

```
LinearWeightedMovingAverage QuantConnect.Algorithm.QCAlgorithm.LWMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new LinearWeightedMovingAverage indicator. This indicator will linearly distribute the weights across the periods.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose LWMA we want.
`Int32`	period	The period over which to compute the LWMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`LinearWeightedMovingAverage` - The new `LinearWeightedMovingAverage` object.

Definition at line 917 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `LinearWeightedMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class LinearWeightedMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.lwma = LinearWeightedMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.lwma.Update(bar.EndTime, bar.Close)

        if self.lwma.IsReady:
            # The current value of self.lwma is represented by self.lwma.Current.Value
            self.Plot("LinearWeightedMovingAverage", "lwma", self.lwma.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class LinearWeightedMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.lwma = LinearWeightedMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.lwma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.lwma.IsReady:
            # The current value of self.lwma is represented by self.lwma.Current.Value
            self.Plot("LinearWeightedMovingAverage", "lwma", self.lwma.Current.Value)
```

The following reference table describes the `LinearWeightedMovingAverage` constructor:

INDICATORS

LinearWeightedMovingAverage() 1/2

```
LinearWeightedMovingAverage QuantConnect.Indicators.LinearWeightedMovingAverage (
string name,
int period
)
```

Initializes a new instance of the LinearWeightedMovingAverage class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the LWMA.

Return

`LinearWeightedMovingAverage` - The new `LinearWeightedMovingAverage` indicator object.

Definition at line 46 of file Indicators/LinearWeightedMovingAverage.cs.

INDICATORS

LinearWeightedMovingAverage() 2/2

```
LinearWeightedMovingAverage QuantConnect.Indicators.LinearWeightedMovingAverage (
int period
)
```

Initializes a new instance of the LinearWeightedMovingAverage class with the default name and period.

Show Details

Parameters		
`int`	period	The period of the LWMA.

Return

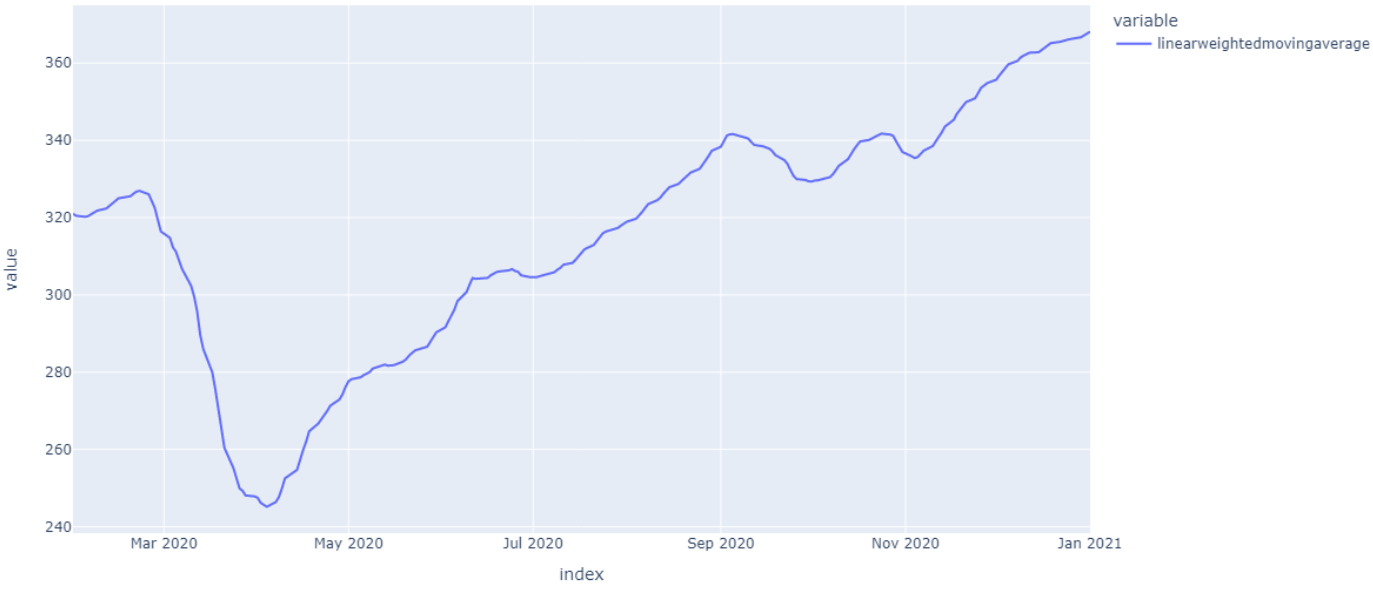
`LinearWeightedMovingAverage` - The new `LinearWeightedMovingAverage` indicator object.

Definition at line 55 of file Indicators/LinearWeightedMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `LinearWeightedMovingAverage` using the plotly library.

LWMA("SPY", 20)


variable
linearweightedmovingaverage
360
340
320
300
280
260
240
value
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Log Return

Introduction

This indicator represents the LogReturn indicator (LOGR) - log returns are useful for identifying price convergence/divergence in a given period - logr = log (current price / last price in period)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using LOGR Indicator

To create an automatic indicators for `LogReturn` , call the `LOGR` helper method from the `QCAlgorithm` class. The `LOGR` method creates a `LogReturn` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class LogReturnAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.logr = self.LOGR(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.logr.IsReady:
            # The current value of self.logr is represented by self.logr.Current.Value
            self.Plot("LogReturn", "logr", self.logr.Current.Value)
```

The following reference table describes the `LOGR` method:

INDICATORS

LOGR() 1/1

```
LogReturn QuantConnect.Algorithm.QCAlgorithm.LOGR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new LogReturn indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose log return we seek.
`Int32`	period	The period of the log return.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`LogReturn` - log return indicator for the requested symbol.

Definition at line 880 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `LogReturn` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class LogReturnAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.logr = LogReturn(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.logr.Update(bar.EndTime, bar.Close)

        if self.logr.IsReady:
            # The current value of self.logr is represented by self.logr.Current.Value
            self.Plot("LogReturn", "logr", self.logr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class LogReturnAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.logr = LogReturn(20)
        self.RegisterIndicator(self.symbol, self.logr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.logr.IsReady:
            # The current value of self.logr is represented by self.logr.Current.Value
            self.Plot("LogReturn", "logr", self.logr.Current.Value)
```

The following reference table describes the `LogReturn` constructor:

INDICATORS

LogReturn() 1/2

```
LogReturn QuantConnect.Indicators.LogReturn (
string name,
int period
)
```

Initializes a new instance of the LogReturn class with the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the LOGR.

Return

`LogReturn` - The new `LogReturn` indicator object.

Definition at line 37 of file Indicators/LogReturn.cs.

INDICATORS

LogReturn() 2/2

```
LogReturn QuantConnect.Indicators.LogReturn (
int period
)
```

Initializes a new instance of the LogReturn class with the default name and period.

Show Details

Parameters		
`int`	period	The period of the SMA.

Return

`LogReturn` - The new `LogReturn` indicator object.

Definition at line 46 of file Indicators/LogReturn.cs.

Visualization

The following image shows plot values of selected properties of `LogReturn` using the plotly library.

Supported Indicators

Mass Index

Introduction

The Mass Index uses the high-low range to identify trend reversals based on range expansions. In this sense, the Mass Index is a volatility indicator that does not have a directional bias. Instead, the Mass Index identifies range bulges that can foreshadow a reversal of the current trend. Developed by Donald Dorsey.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MASS Indicator

To create an automatic indicators for `MassIndex` , call the `MASS` helper method from the `QCAlgorithm` class. The `MASS` method creates a `MassIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class MassIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mass = self.MASS(self.symbol, 9, 25)

    def OnData(self, slice: Slice) -> None:
        if self.mass.IsReady:
            # The current value of self.mass is represented by self.mass.Current.Value
            self.Plot("MassIndex", "mass", self.mass.Current.Value)
```

The following reference table describes the `MASS` method:

INDICATORS

MASS() 1/1

```
MassIndex QuantConnect.Algorithm.QCAlgorithm.MASS (
 Symbol symbol,
 *Int32 emaPeriod,
 *Int32 sumPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new Mass Index indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Mass Index we want.
`*Int32`	emaPeriod	*(Optional)* The period used by both EMA.
`*Int32`	sumPeriod	*(Optional)* The sum period.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`MassIndex` - The Mass Index indicator for the requested symbol over the specified period.

Definition at line 1073 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MassIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MassIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mass = MassIndex(9, 25)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.mass.Update(bar)

        if self.mass.IsReady:
            # The current value of self.mass is represented by self.mass.Current.Value
            self.Plot("MassIndex", "mass", self.mass.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MassIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mass = MassIndex(9, 25)
        self.RegisterIndicator(self.symbol, self.mass, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.mass.IsReady:
            # The current value of self.mass is represented by self.mass.Current.Value
            self.Plot("MassIndex", "mass", self.mass.Current.Value)
```

The following reference table describes the `MassIndex` constructor:

INDICATORS

MassIndex() 1/2

```
MassIndex QuantConnect.Indicators.MassIndex (
string name,
int emaPeriod,
int sumPeriod
)
```

Initializes a new instance of the `MassInde` class.

Show Details

Parameters		
`string`	name	The name for this instance.
`int`	emaPeriod	The period used by both EMA.
`int`	sumPeriod	The sum period.

Return

`MassIndex` - The new `MassIndex` indicator object.

Definition at line 39 of file Indicators/MassIndex.cs.

INDICATORS

MassIndex() 2/2

```
MassIndex QuantConnect.Indicators.MassIndex (
*int emaPeriod,
*int sumPeriod
)
```

Initializes a new instance of the MassInde class.

Show Details

Parameters		
*int	emaPeriod	*(Optional) (Optional)* The period used by both EMA. Default: 9.
*int	sumPeriod	*(Optional) (Optional)* The sum period. Default: 25.

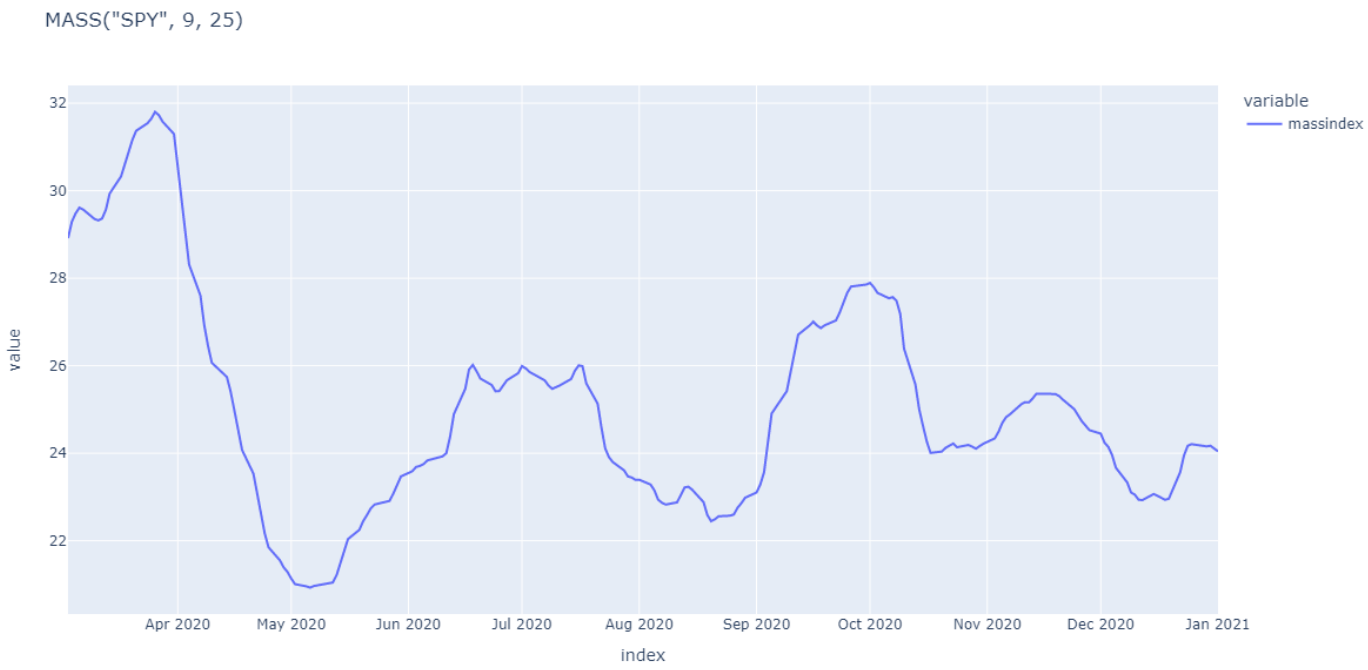
Return

MassIndex - The new MassIndex indicator object.

Definition at line 53 of file Indicators/MassIndex.cs.

Visualization

The following image shows plot values of selected properties of MassIndex using the plotly library.



Supported Indicators

Maximum

Introduction

This indicator represents an indicator capable of tracking the maximum value and how many periods ago it occurred

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MAX Indicator

To create an automatic indicators for `Maximum` , call the `MAX` helper method from the `QCAlgorithm` class. The `MAX` method creates a `Maximum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MaximumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.max = self.MAX(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.max.IsReady:
            # The current value of self.max is represented by self.max.Current.Value
            self.Plot("Maximum", "max", self.max.Current.Value)
            # Plot all attributes of self.max
            self.Plot("Maximum", "periodssincemaximum", self.max.PeriodsSinceMaximum.Current.Value)
```

The following reference table describes the `MAX` method:

INDICATORS

MAX() 1/1

```
Maximum QuantConnect.Algorithm.QCAlgorithm.MAX (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Maximum indicator to compute the maximum value.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose max we want.
`Int32`	period	The look back period over which to compute the max value.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None and the symbol is of type TradeBar defaults to the High property, otherwise it defaults to Value property of BaseData (x => x.Value).

Return

`Maximum` - A Maximum indicator that compute the max value and the periods since the max value.

Definition at line 1017 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Maximum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class MaximumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.max = Maximum(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.max.Update(bar.EndTime, bar.Close)

        if self.max.IsReady:
            # The current value of self.max is represented by self.max.Current.Value
            self.Plot("Maximum", "max", self.max.Current.Value)
            # Plot all attributes of self.max
            self.Plot("Maximum", "periodssincemaximum", self.max.PeriodsSinceMaximum.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MaximumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.max = Maximum(20)
        self.RegisterIndicator(self.symbol, self.max, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.max.IsReady:
            # The current value of self.max is represented by self.max.Current.Value
            self.Plot("Maximum", "max", self.max.Current.Value)
            # Plot all attributes of self.max
            self.Plot("Maximum", "periodssincemaximum", self.max.PeriodsSinceMaximum.Current.Value)
```

The following reference table describes the `Maximum` constructor:

INDICATORS

Maximum() 1/2

```
Maximum QuantConnect.Indicators.Maximum (
int period
)
```

Creates a new Maximum indicator with the specified period.

Show Details

Parameters		
int	period	The period over which to look back.

Return

`Maximum` - The new `Maximum` indicator object.

Definition at line 44 of file Indicators/Maximum.cs.

INDICATORS

Maximum() 2/2

```
Maximum QuantConnect.Indicators.Maximum (
string name,
int period
)
```

Creates a new Maximum indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period over which to look back.

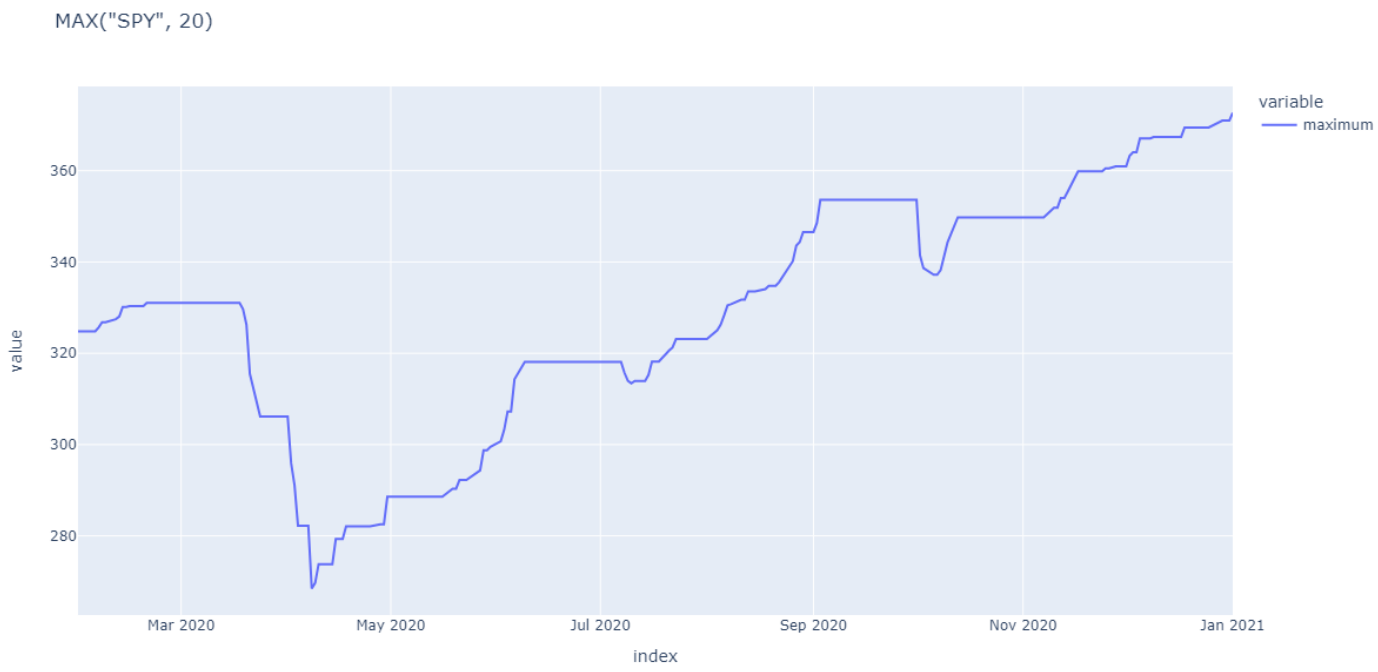
Return

`Maximum` - The new `Maximum` indicator object.

Definition at line 54 of file Indicators/Maximum.cs.

Visualization

The following image shows plot values of selected properties of `Maximum` using the plotly library.



Supported Indicators

Mc Clellan Oscillator

Introduction

The McClellan Oscillator is a market breadth indicator which was developed by Sherman and Marian McClellan. It is based on the difference between the number of advancing and declining periods.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOSC Indicator

To create an automatic indicators for `McClellanOscillator` , call the `MOSC` helper method from the `QCAlgorithm` class. The `MOSC` method creates a `McClellanOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class McClellanOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mosc = self.MOSC([self.symbol, Symbol.Create("QQQ", SecurityType.Equity, Market.USA)])

    def OnData(self, slice: Slice) -> None:
        if self.mosc.IsReady:
            # The current value of self.mosc is represented by self.mosc.Current.Value
            self.Plot("McClellanOscillator", "mosc", self.mosc.Current.Value)
            # Plot all attributes of self.mosc
            self.Plot("McClellanOscillator", "emafast", self.mosc.EMAFast.Current.Value)
            self.Plot("McClellanOscillator", "emaslow", self.mosc.EMASlow.Current.Value)
            self.Plot("McClellanOscillator", "addifference", self.mosc.ADDifference.Current.Value)
```

The following reference table describes the `MOSC` method:

INDICATORS

MOSC() 1/2

```
McClellanOscillator QuantConnect.Algorithm.QCAlgorithm.MOSC (
IEnumerable<Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Oscillator indicator.

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`*Int32`	fastPeriod	*(Optional)* /
`*Int32`	slowPeriod	*(Optional)* /
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* /

Return

`McClellanOscillator` - The new `McClellanOscillator` object.

Definition at line 2044 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MOSC() 2/2

```
McClellanOscillator QuantConnect.Algorithm.QCAlgorithm.MOSC (
Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Oscillator indicator.

Show Details

Parameters		
`Symbol>`	symbols	The symbols whose McClellan Oscillator we want.
`*Int32`	fastPeriod	*(Optional)* Fast period EMA of advance decline difference.
`*Int32`	slowPeriod	*(Optional)* Slow period EMA of advance decline difference.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`McClellanOscillator` - The McClellan Oscillator indicator for the requested symbol over the specified period.

Definition at line 2059 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `McClellanOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class McClellanOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mosc = McClellanOscillator("")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.mosc.Update(bar)

        if self.mosc.IsReady:
            # The current value of self.mosc is represented by self.mosc.Current.Value
            self.Plot("McClellanOscillator", "mosc", self.mosc.Current.Value)
            # Plot all attributes of self.mosc
            self.Plot("McClellanOscillator", "emafast", self.mosc.EMAFast.Current.Value)
            self.Plot("McClellanOscillator", "emaslow", self.mosc.EMASlow.Current.Value)
            self.Plot("McClellanOscillator", "addifference", self.mosc.ADDifference.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class McClellanOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mosc = McClellanOscillator("")
        self.RegisterIndicator(self.symbol, self.mosc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.mosc.IsReady:
            # The current value of self.mosc is represented by self.mosc.Current.Value
            self.Plot("McClellanOscillator", "mosc", self.mosc.Current.Value)
            # Plot all attributes of self.mosc
            self.Plot("McClellanOscillator", "emafast", self.mosc.EMAFast.Current.Value)
            self.Plot("McClellanOscillator", "emaslow", self.mosc.EMASlow.Current.Value)
            self.Plot("McClellanOscillator", "addifference", self.mosc.ADDifference.Current.Value)
```

The following reference table describes the `McClellanOscillator` constructor:

INDICATORS

McClellanOscillator() 1/2

```
McClellanOscillator QuantConnect.Indicators.McClellanOscillator (
string name,
*int fastPeriod,
*int slowPeriod
)
```

The slow period of EMA of advance decline difference.

Show Details

Parameters		
`string`	name	The name of the indicator.
`*int`	fastPeriod	*(Optional) (Optional)* The fast period of EMA of advance decline difference. Default: 19.
`*int`	slowPeriod	*(Optional) (Optional)* The slow period of EMA of advance decline difference. Default: 39.

Return

`McClellanOscillator` - The new `McClellanOscillator` indicator object.

Definition at line 61 of file Indicators/McClellanOscillator.cs.

INDICATORS

McClellanOscillator() 2/2

```
McClellanOscillator QuantConnect.Indicators.McClellanOscillator (
*int fastPeriod,
*int slowPeriod
)
```

The slow period of EMA of advance decline difference.

Show Details

Parameters		
*int	fastPeriod	*(Optional) (Optional)* The fast period of EMA of advance decline difference. Default: 19.
*int	slowPeriod	*(Optional) (Optional)* The slow period of EMA of advance decline difference. Default: 39.

Return

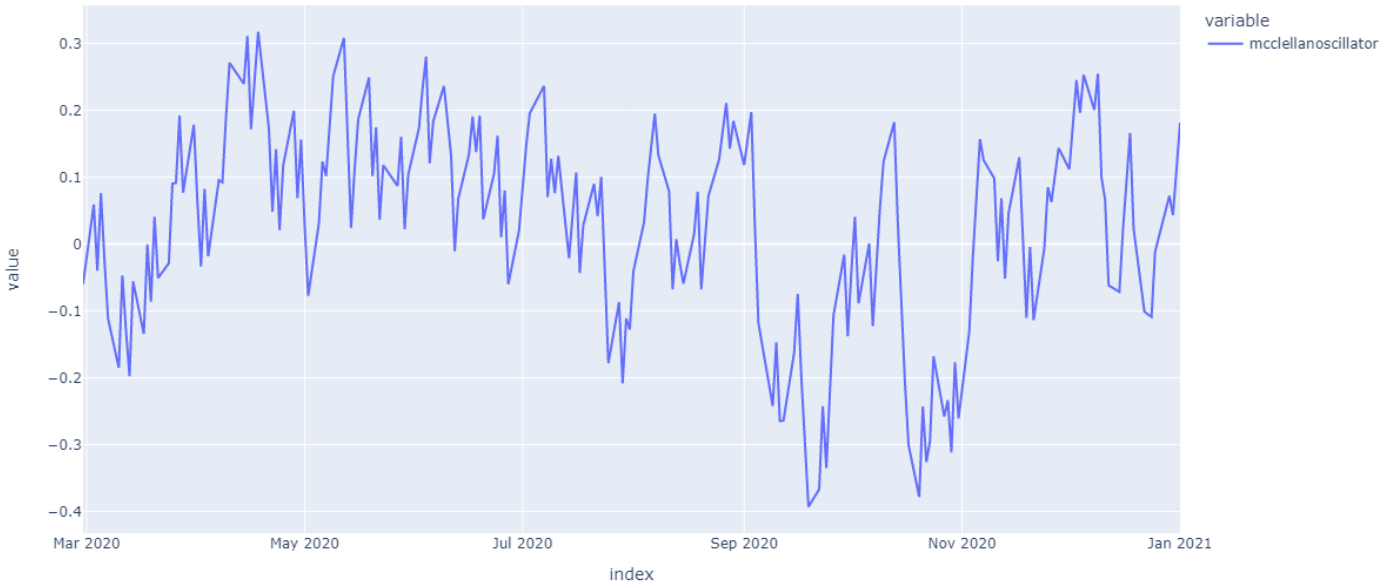
`McClellanOscillator` - The new `McClellanOscillator` indicator object.

Definition at line 79 of file Indicators/McClellanOscillator.cs.

Visualization

The following image shows plot values of selected properties of `McClellanOscillator` using the plotly library.

MOSC(["SPY", "QQQ"])


variable
mcclellanoscillator
value
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Mc Clellan Summation Index

Introduction

The McClellan Summation Index (MSI) is a market breadth indicator that is based on the rolling average of difference between the number of advancing and declining issues on a stock exchange. It is generally considered as is a long-term version of the

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MSI Indicator

To create an automatic indicators for `McClellanSummationIndex` , call the `MSI` helper method from the `QCAlgorithm` class. The `MSI` method creates a `McClellanSummationIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class McClellanSummationIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.msi = self.MSI([self.symbol, Symbol.Create("QQQ", SecurityType.Equity, Market.USA)])

    def OnData(self, slice: Slice) -> None:
        if self.msi.IsReady:
            # The current value of self.msi is represented by self.msi.Current.Value
            self.Plot("McClellanSummationIndex", "msi", self.msi.Current.Value)
            # Plot all attributes of self.msi
            self.Plot("McClellanSummationIndex", "mcclellanoscillator",
self.msi.McClellanOscillator.Current.Value)
```

The following reference table describes the `MSI` method:

INDICATORS

MSI() 1/2

```
McClellanSummationIndex QuantConnect.Algorithm.QCAlgorithm.MSI (
IEnumerable<Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Summation Index indicator.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
*Int32	fastPeriod	*(Optional)* /
*Int32	slowPeriod	*(Optional)* /
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, TradeBar>	selector	*(Optional)* /

Return

`McClellanSummationIndex` - The new `McClellanSummationIndex` object.

Definition at line 2082 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MSI() 2/2

```
McClellanSummationIndex QuantConnect.Algorithm.QCAlgorithm.MSI (
Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Summation Index indicator.

Show Details

Parameters		
`Symbol>`	symbols	The symbols whose McClellan Summation Index we want.
`*Int32`	fastPeriod	*(Optional)* Fast period EMA of advance decline difference.
`*Int32`	slowPeriod	*(Optional)* Slow period EMA of advance decline difference.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`McClellanSummationIndex` - The McClellan Summation Index indicator for the requested symbol over the specified period.

Definition at line 2097 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `McClellanSummationIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class McClellanSummationIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.msi = McClellanSummationIndex("")

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.msi.Update(bar)

        if self.msi.IsReady:
            # The current value of self.msi is represented by self.msi.Current.Value
            self.Plot("McClellanSummationIndex", "msi", self.msi.Current.Value)
            # Plot all attributes of self.msi
            self.Plot("McClellanSummationIndex", "mcclellanoscillator",
self.msi.McClellanOscillator.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class McClellanSummationIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.msi = McClellanSummationIndex("")
        self.RegisterIndicator(self.symbol, self.msi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.msi.IsReady:
            # The current value of self.msi is represented by self.msi.Current.Value
            self.Plot("McClellanSummationIndex", "msi", self.msi.Current.Value)
            # Plot all attributes of self.msi
            self.Plot("McClellanSummationIndex", "mcclellanoscillator",
self.msi.McClellanOscillator.Current.Value)
```

The following reference table describes the `McClellanSummationIndex` constructor:

INDICATORS

McClellanSummationIndex() 1/2

```
McClellanSummationIndex QuantConnect.Indicators.McClellanSummationIndex (
string name,
*int fastPeriod,
*int slowPeriod
)
```

The slow period of EMA of advance decline difference.

Show Details

Parameters		
`string`	name	The name of the indicator.
`*int`	fastPeriod	*(Optional) (Optional)* The fast period of EMA of advance decline difference. Default: 19.
`*int`	slowPeriod	*(Optional) (Optional)* The slow period of EMA of advance decline difference. Default: 39.

Return

`McClellanSummationIndex` - The new `McClellanSummationIndex` indicator object.

Definition at line 54 of file Indicators/McClellanSummationIndex.cs.

INDICATORS

McClellanSummationIndex() 2/2

```
McClellanSummationIndex QuantConnect.Indicators.McClellanSummationIndex (
*int fastPeriod,
*int slowPeriod
)
```

The slow period of EMA of advance decline difference.

Show Details

Parameters		
*int	fastPeriod	*(Optional) (Optional)* The fast period of EMA of advance decline difference. Default: 19.
*int	slowPeriod	*(Optional) (Optional)* The slow period of EMA of advance decline difference. Default: 39.

Return

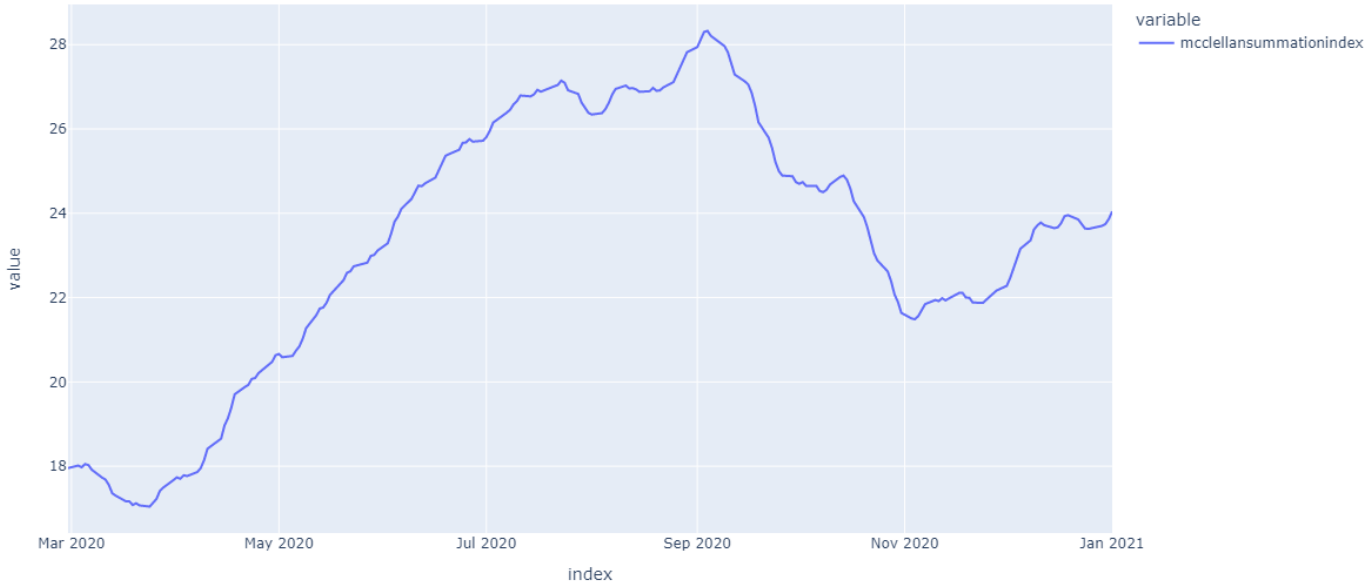
`McClellanSummationIndex` - The new `McClellanSummationIndex` indicator object.

Definition at line 74 of file Indicators/McClellanSummationIndex.cs.

Visualization

The following image shows plot values of selected properties of `McClellanSummationIndex` using the plotly library.

MSI(["SPY", "QQQ"])


variable
mcclellansummationindex
28
26
24
22
20
18
value
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Mean Absolute Deviation

Introduction

This indicator computes the n-period mean absolute deviation.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MAD Indicator

To create an automatic indicators for `MeanAbsoluteDeviation` , call the `MAD` helper method from the `QCAlgorithm` class. The `MAD` method creates a `MeanAbsoluteDeviation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MeanAbsoluteDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mad = self.MAD(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.mad.IsReady:
            # The current value of self.mad is represented by self.mad.Current.Value
            self.Plot("MeanAbsoluteDeviation", "mad", self.mad.Current.Value)
            # Plot all attributes of self.mad
            self.Plot("MeanAbsoluteDeviation", "mad", self.mad.Mean.Current.Value)
```

The following reference table describes the `MAD` method:

INDICATORS

MAD() 1/1

```
MeanAbsoluteDeviation QuantConnect.Algorithm.QCAlgorithm.MAD (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new MeanAbsoluteDeviation indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MeanAbsoluteDeviation we want.
`Int32`	period	The period over which to compute the MeanAbsoluteDeviation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MeanAbsoluteDeviation` - The MeanAbsoluteDeviation indicator for the requested symbol over the specified period.

Definition at line 956 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MeanAbsoluteDeviation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class MeanAbsoluteDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mad = MeanAbsoluteDeviation(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.mad.Update(bar.EndTime, bar.Close)

        if self.mad.IsReady:
            # The current value of self.mad is represented by self.mad.Current.Value
            self.Plot("MeanAbsoluteDeviation", "mad", self.mad.Current.Value)
            # Plot all attributes of self.mad
            self.Plot("MeanAbsoluteDeviation", "mean", self.mad.Mean.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MeanAbsoluteDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mad = MeanAbsoluteDeviation(20)
        self.RegisterIndicator(self.symbol, self.mad, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.mad.IsReady:
            # The current value of self.mad is represented by self.mad.Current.Value
            self.Plot("MeanAbsoluteDeviation", "mad", self.mad.Current.Value)
            # Plot all attributes of self.mad
            self.Plot("MeanAbsoluteDeviation", "mean", self.mad.Mean.Current.Value)
```

The following reference table describes the `MeanAbsoluteDeviation` constructor:

INDICATORS

MeanAbsoluteDeviation() 1/2

```
MeanAbsoluteDeviation QuantConnect.Indicators.MeanAbsoluteDeviation (
int period
)
```

Evaluates the mean absolute deviation of samples in the lookback period.

Show Details

Parameters		
int	period	The sample size of the standard deviation.

Return

`MeanAbsoluteDeviation` - The new `MeanAbsoluteDeviation` indicator object.

Definition at line 37 of file Indicators/MeanAbsoluteDeviation.cs.

INDICATORS

MeanAbsoluteDeviation() 2/2

```
MeanAbsoluteDeviation QuantConnect.Indicators.MeanAbsoluteDeviation (
string name,
int period
)
```

Evaluates the mean absolute deviation of samples in the look-back period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The sample size of the mean absolute deviation.

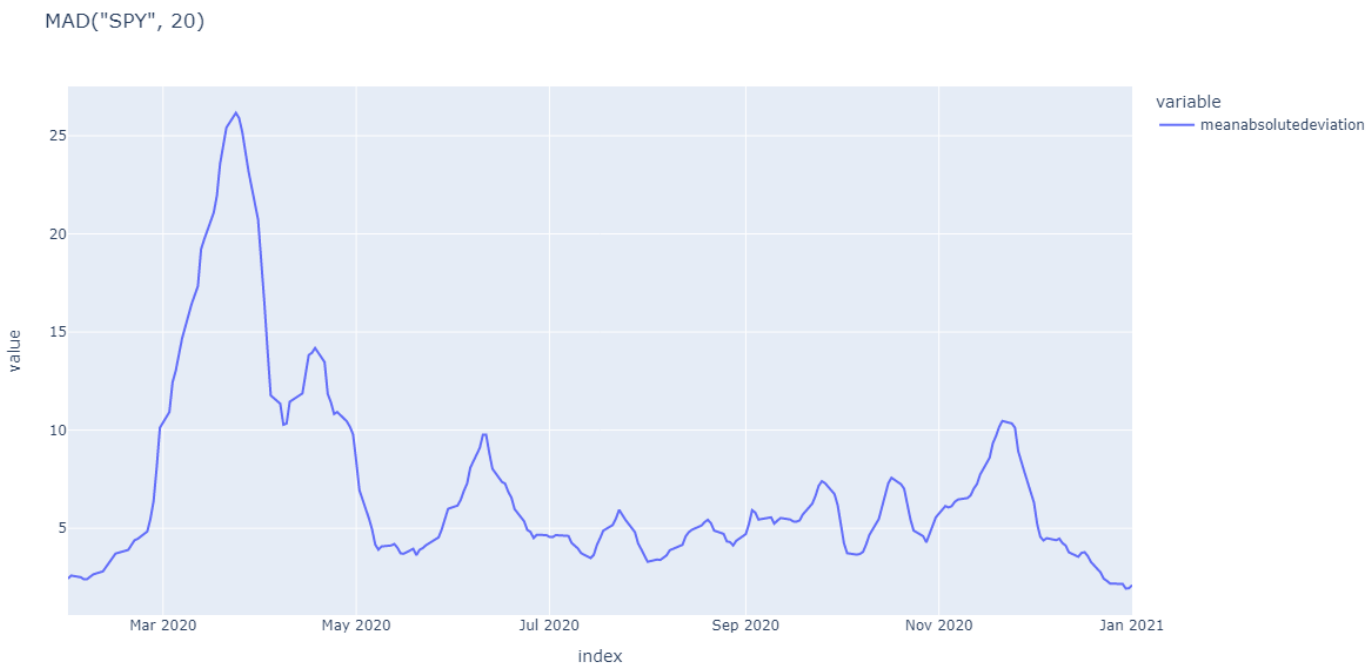
Return

`MeanAbsoluteDeviation` - The new `MeanAbsoluteDeviation` indicator object.

Definition at line 49 of file Indicators/MeanAbsoluteDeviation.cs.

Visualization

The following image shows plot values of selected properties of `MeanAbsoluteDeviation` using the plotly library.



Supported Indicators

Mid Point

Introduction

This indicator computes the MidPoint (MIDPOINT) The MidPoint is calculated using the following formula: MIDPOINT = (Highest Value + Lowest Value) / 2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MIDPOINT Indicator

To create an automatic indicators for `MidPoint` , call the `MIDPOINT` helper method from the `QCAlgorithm` class. The `MIDPOINT` method creates a `MidPoint` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class MidPointAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.midpoint = self.MIDPOINT(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.midpoint.IsReady:
            # The current value of self.midpoint is represented by self.midpoint.Current.Value
            self.Plot("MidPoint", "midpoint", self.midpoint.Current.Value)
```

The following reference table describes the `MIDPOINT` method:

INDICATORS

MIDPOINT() 1/1

```
MidPoint QuantConnect.Algorithm.QCAlgorithm.MIDPOINT (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new MidPoint indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MIDPOINT we want.
`Int32`	period	The period over which to compute the MIDPOINT.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MidPoint` - The MidPoint indicator for the requested symbol over the specified period.

Definition at line 1091 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MidPoint` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```
PY
class MidPointAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.midpoint = MidPoint(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.midpoint.Update(bar.EndTime, bar.Close)

        if self.midpoint.IsReady:
            # The current value of self.midpoint is represented by self.midpoint.Current.Value
            self.Plot("MidPoint", "midpoint", self.midpoint.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MidPointAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.midpoint = MidPoint(20)
        self.RegisterIndicator(self.symbol, self.midpoint, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.midpoint.IsReady:
            # The current value of self.midpoint is represented by self.midpoint.Current.Value
            self.Plot("MidPoint", "midpoint", self.midpoint.Current.Value)
```

The following reference table describes the `MidPoint` constructor:

INDICATORS

MidPoint() 1/2

```
MidPoint QuantConnect.Indicators.MidPoint (
string name,
int period
)
```

Initializes a new instance of the `MidPoin` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the MIDPOINT.

Return

`MidPoint` - The new `MidPoint` indicator object.

Definition at line 34 of file Indicators/MidPoint.cs.

INDICATORS

MidPoint() 2/2

```
MidPoint QuantConnect.Indicators.MidPoint (
int period
)
```

Initializes a new instance of the `MidPoin` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the MIDPOINT.

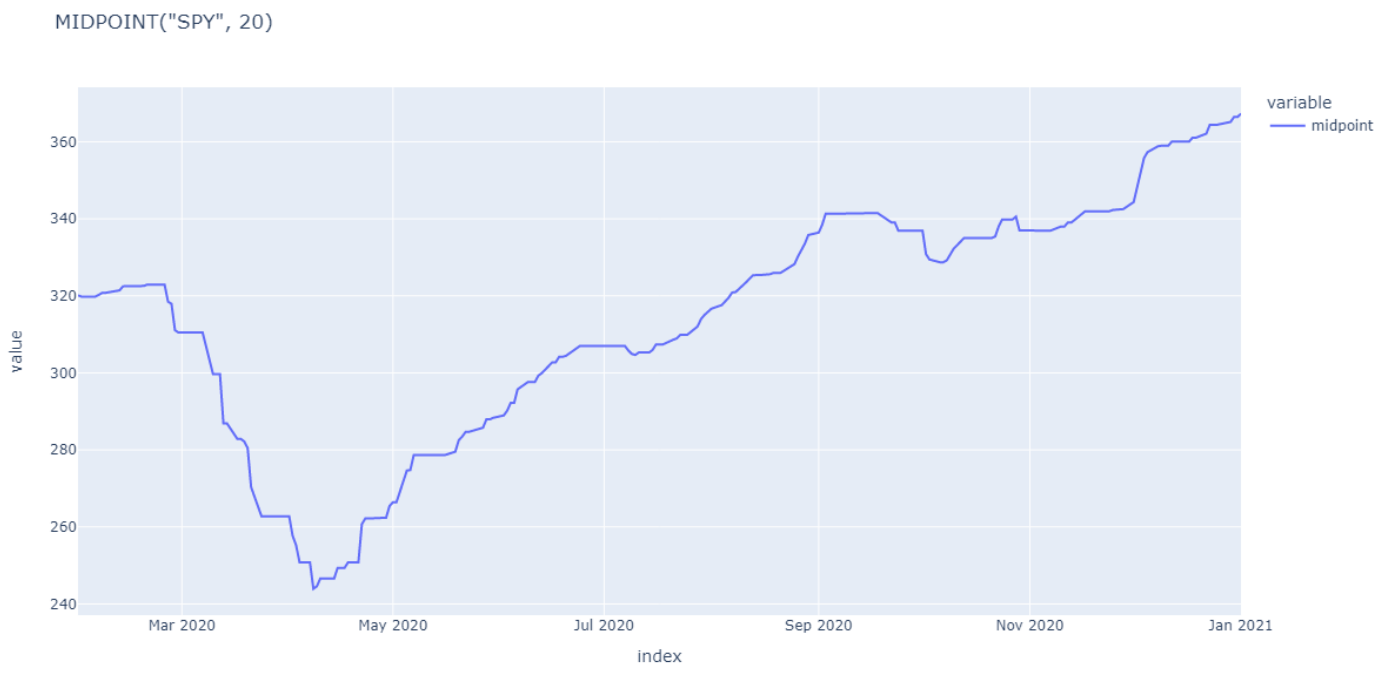
Return

`MidPoint` - The new `MidPoint` indicator object.

Definition at line 46 of file Indicators/MidPoint.cs.

Visualization

The following image shows plot values of selected properties of `MidPoint` using the plotly library.



Supported Indicators

Mid Price

Introduction

This indicator computes the MidPrice (MIDPRICE). The MidPrice is calculated using the following formula: MIDPRICE = (Highest High + Lowest Low) / 2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MIDPRICE Indicator

To create an automatic indicators for `MidPrice` , call the `MIDPRICE` helper method from the `QCAlgorithm` class. The `MIDPRICE` method creates a `MidPrice` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class MidPriceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.midprice = self.MIDPRICE(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.midprice.IsReady:
            # The current value of self.midprice is represented by self.midprice.Current.Value
            self.Plot("MidPrice", "midprice", self.midprice.Current.Value)
```

The following reference table describes the `MIDPRICE` method:

INDICATORS

MIDPRICE() 1/1

```
MidPrice QuantConnect.Algorithm.QCAlgorithm.MIDPRICE (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new MidPrice indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MIDPRICE we want.
`Int32`	period	The period over which to compute the MIDPRICE.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`MidPrice` - The MidPrice indicator for the requested symbol over the specified period.

Definition at line 1109 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MidPrice` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class MidPriceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.midprice = MidPrice(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.midprice.Update(bar)

        if self.midprice.IsReady:
            # The current value of self.midprice is represented by self.midprice.Current.Value
            self.Plot("MidPrice", "midprice", self.midprice.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MidPriceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.midprice = MidPrice(20)
        self.RegisterIndicator(self.symbol, self.midprice, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.midprice.IsReady:
            # The current value of self.midprice is represented by self.midprice.Current.Value
            self.Plot("MidPrice", "midprice", self.midprice.Current.Value)
```

The following reference table describes the `MidPrice` constructor:

INDICATORS

MidPrice() 1/2

```
MidPrice QuantConnect.Indicators.MidPrice (
string name,
int period
)
```

Initializes a new instance of the `MidPric` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the MIDPRICE.

Return

`MidPrice` - The new `MidPrice` indicator object.

Definition at line 36 of file Indicators/MidPrice.cs.

INDICATORS

MidPrice() 2/2

```
MidPrice QuantConnect.Indicators.MidPrice (
int period
)
```

Initializes a new instance of the `MidPric` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the MIDPRICE.

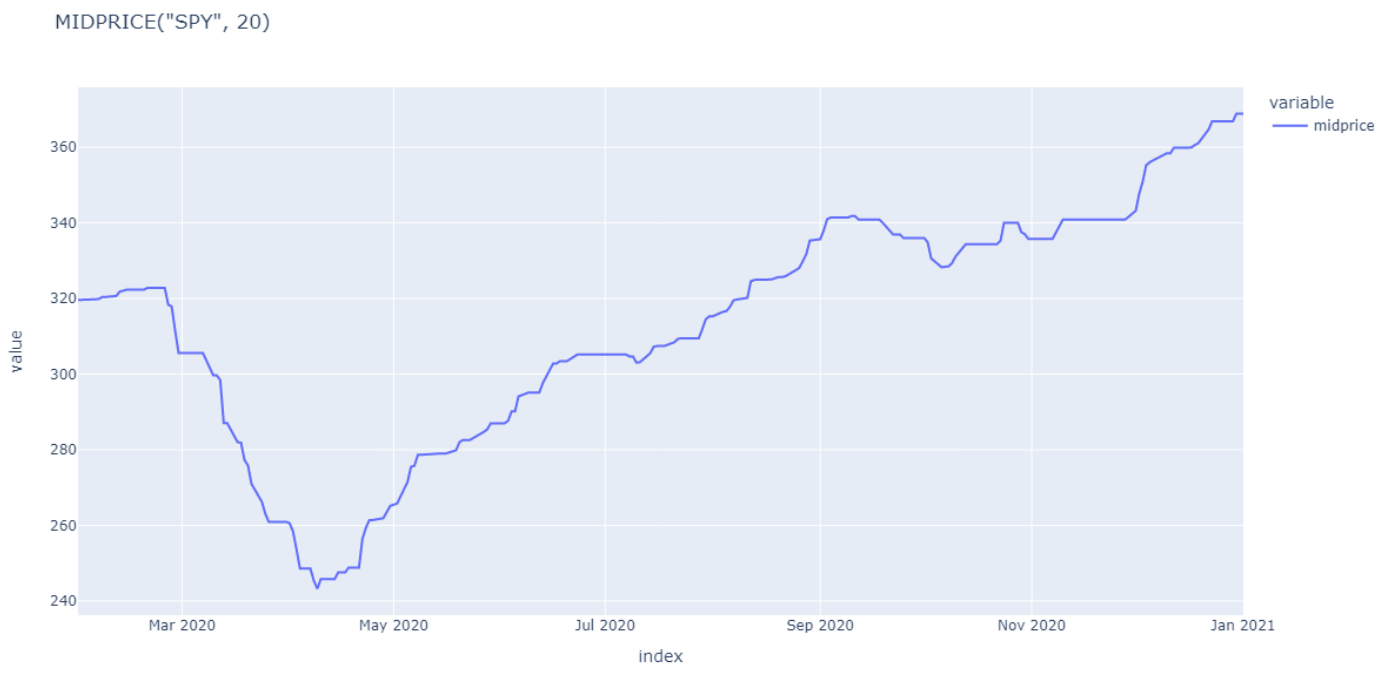
Return

`MidPrice` - The new `MidPrice` indicator object.

Definition at line 48 of file Indicators/MidPrice.cs.

Visualization

The following image shows plot values of selected properties of `MidPrice` using the plotly library.



Supported Indicators

Minimum

Introduction

This indicator represents an indicator capable of tracking the minimum value and how many periods ago it occurred

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MIN Indicator

To create an automatic indicators for `Minimum` , call the `MIN` helper method from the `QCAlgorithm` class. The `MIN` method creates a `Minimum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MinimumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.min = self.MIN(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.min.IsReady:
            # The current value of self.min is represented by self.min.Current.Value
            self.Plot("Minimum", "min", self.min.Current.Value)
            # Plot all attributes of self.min
            self.Plot("Minimum", "periodssinceminimum", self.min.PeriodsSinceMinimum.Current.Value)
```

The following reference table describes the `MIN` method:

INDICATORS

MIN() 1/1

```
Minimum QuantConnect.Algorithm.QCAlgorithm.MIN (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new Minimum indicator to compute the minimum value.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose min we want.
`Int32`	period	The look back period over which to compute the min value.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None and the symbol is of type TradeBar defaults to the Low property, otherwise it defaults to Value property of BaseData (x => x.Value).

Return

`Minimum` - A Minimum indicator that compute the in value and the periods since the min value.

Definition at line 1128 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Minimum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class MinimumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.min = Minimum(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.min.Update(bar.EndTime, bar.Close)

        if self.min.IsReady:
            # The current value of self.min is represented by self.min.Current.Value
            self.Plot("Minimum", "min", self.min.Current.Value)
            # Plot all attributes of self.min
            self.Plot("Minimum", "periodssinceminimum", self.min.PeriodsSinceMinimum.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MinimumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.min = Minimum(20)
        self.RegisterIndicator(self.symbol, self.min, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.min.IsReady:
            # The current value of self.min is represented by self.min.Current.Value
            self.Plot("Minimum", "min", self.min.Current.Value)
            # Plot all attributes of self.min
            self.Plot("Minimum", "periodssinceminimum", self.min.PeriodsSinceMinimum.Current.Value)
```

The following reference table describes the `Minimum` constructor:

INDICATORS

Minimum() 1/2

```
Minimum QuantConnect.Indicators.Minimum (
int period
)
```

Creates a new Minimum indicator with the specified period.

Show Details

Parameters		
int	period	The period over which to look back.

Return

`Minimum` - The new `Minimum` indicator object.

Definition at line 44 of file Indicators/Minimum.cs.

INDICATORS

Minimum() 2/2

```
Minimum QuantConnect.Indicators.Minimum (
string name,
int period
)
```

Creates a new Minimum indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period over which to look back.

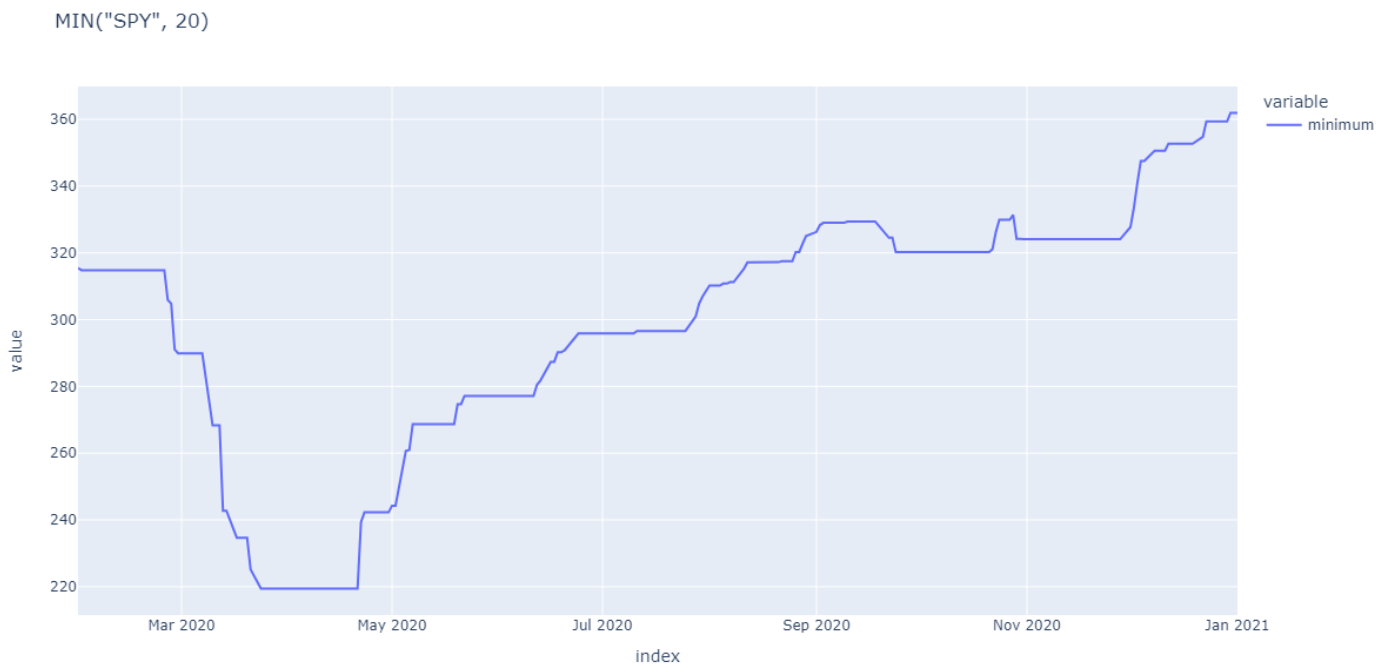
Return

`Minimum` - The new `Minimum` indicator object.

Definition at line 54 of file Indicators/Minimum.cs.

Visualization

The following image shows plot values of selected properties of `Minimum` using the plotly library.



Supported Indicators

Momentum

Introduction

This indicator computes the n-period change in a value using the following: value_0 - value_n

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOM Indicator

To create an automatic indicators for `Momentum` , call the `MOM` helper method from the `QCAlgorithm` class. The `MOM` method creates a `Momentum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MomentumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mom = self.MOM(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.mom.IsReady:
            # The current value of self.mom is represented by self.mom.Current.Value
            self.Plot("Momentum", "mom", self.mom.Current.Value)
```

The following reference table describes the `MOM` method:

INDICATORS

MOM() 1/1

```
Momentum QuantConnect.Algorithm.QCAlgorithm.MOM (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Momentum indicator. This will compute the absolute n-period change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose momentum we want.
`Int32`	period	The period over which to compute the momentum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Momentum` - The momentum indicator for the requested symbol over the specified period.

Definition at line 1164 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Momentum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class MomentumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mom = Momentum(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.mom.Update(bar.EndTime, bar.Close)

        if self.mom.IsReady:
            # The current value of self.mom is represented by self.mom.Current.Value
            self.Plot("Momentum", "mom", self.mom.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class MomentumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mom = Momentum(20)
        self.RegisterIndicator(self.symbol, self.mom, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.mom.IsReady:
            # The current value of self.mom is represented by self.mom.Current.Value
            self.Plot("Momentum", "mom", self.mom.Current.Value)
```

The following reference table describes the `Momentum` constructor:

INDICATORS

Momentum() 1/2

```
Momentum QuantConnect.Indicators.Momentum (
int period
)
```

Creates a new Momentum indicator with the specified period.

Show Details

Parameters		
int	period	The period over which to perform to computation.

Return

`Momentum` - The new `Momentum` indicator object.

Definition at line 33 of file Indicators/Momentum.cs.

INDICATORS

Momentum() 2/2

```
Momentum QuantConnect.Indicators.Momentum (
string name,
int period
)
```

Creates a new Momentum indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period over which to perform to computation.

Return

`Momentum` - The new `Momentum` indicator object.

Definition at line 43 of file Indicators/Momentum.cs.

Visualization

The following image shows plot values of selected properties of `Momentum` using the plotly library.

Supported Indicators

Momentum Percent

Introduction

This indicator computes the n-period percentage rate of change in a value using the following: 100 * (value_0 - value_n) / value_n This indicator yields the same results of RateOfChangePercent

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOMP Indicator

To create an automatic indicators for `MomentumPercent` , call the `MOMP` helper method from the `QCAlgorithm` class. The `MOMP` method creates a `MomentumPercent` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class MomentumPercentAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.momp = self.MOMP(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.momp.IsReady:
            # The current value of self.momp is represented by self.momp.Current.Value
            self.Plot("MomentumPercent", "momp", self.momp.Current.Value)
```

The following reference table describes the `MOMP` method:

INDICATORS

MOMP() 1/1

```
MomentumPercent QuantConnect.Algorithm.QCAlgorithm.MOMP (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new MomentumPercent indicator. This will compute the n-period percent change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose momentum we want.
`Int32`	period	The period over which to compute the momentum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MomentumPercent` - The momentum indicator for the requested symbol over the specified period.

Definition at line 1202 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MomentumPercent` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class MomentumPercentAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.momp = MomentumPercent(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.momp.Update(bar.EndTime, bar.Close)

        if self.momp.IsReady:
            # The current value of self.momp is represented by self.momp.Current.Value
            self.Plot("MomentumPercent", "momp", self.momp.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class MomentumPercentAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.momp = MomentumPercent(20)
        self.RegisterIndicator(self.symbol, self.momp, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.momp.IsReady:
            # The current value of self.momp is represented by self.momp.Current.Value
            self.Plot("MomentumPercent", "momp", self.momp.Current.Value)
```

The following reference table describes the `MomentumPercent` constructor:

INDICATORS

MomentumPercent() 1/2

```
MomentumPercent QuantConnect.Indicators.MomentumPercent (
int period
)
```

Creates a new MomentumPercent indicator with the specified period.

Show Details

Parameters		
int	period	The period over which to perform to computation.

Return

`MomentumPercent` - The new `MomentumPercent` indicator object.

Definition at line 30 of file Indicators/MomentumPercent.cs.

INDICATORS

MomentumPercent() 2/2

```
MomentumPercent QuantConnect.Indicators.MomentumPercent (
string name,
int period
)
```

Creates a new MomentumPercent indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period over which to perform to computation.

Return

`MomentumPercent` - The new `MomentumPercent` indicator object.

Definition at line 40 of file Indicators/MomentumPercent.cs.

Visualization

The following image shows plot values of selected properties of `MomentumPercent` using the plotly library.

Supported Indicators

Momersion Indicator

Introduction

Oscillator indicator that measures momentum and mean-reversion over a specified period n. source Harris, Michael. "Momersion Indicator." Price Action Lab., 13 Aug. 2015. Web.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOMERSION Indicator

To create an automatic indicators for `MomersionIndicator` , call the `MOMERSION` helper method from the `QCAlgorithm` class. The `MOMERSION` method creates a `MomersionIndicator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class MomersionIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.momersion = self.MOMERSION(self.symbol, 10, 20)

    def OnData(self, slice: Slice) -> None:
        if self.momersion.IsReady:
            # The current value of self.momersion is represented by self.momersion.Current.Value
            self.Plot("MomersionIndicator", "momersion", self.momersion.Current.Value)
```

The following reference table describes the `MOMERSION` method:

INDICATORS

MOMERSION() 1/1

```
MomersionIndicator QuantConnect.Algorithm.QCAlgorithm.MOMERSION (
Symbol symbol,
Nullable<Int32> minPeriod,
Int32 fullPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Momersion indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Momersion we want.
`Nullable<Int32>`	minPeriod	The minimum period over which to compute the Momersion. Must be greater than 3. If None, only full period will be used in computations.
`Int32`	fullPeriod	The full period over which to compute the Momersion.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MomersionIndicator` - The Momersion indicator for the requested symbol over the specified period.

Definition at line 1183 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MomersionIndicator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class MomersionIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.momersion = MomersionIndicator(10, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.momersion.Update(bar.EndTime, bar.Close)

        if self.momersion.IsReady:
            # The current value of self.momersion is represented by self.momersion.Current.Value
            self.Plot("MomersionIndicator", "momersion", self.momersion.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MomersionIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.momersion = MomersionIndicator(10, 20)
        self.RegisterIndicator(self.symbol, self.momersion, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.momersion.IsReady:
            # The current value of self.momersion is represented by self.momersion.Current.Value
            self.Plot("MomersionIndicator", "momersion", self.momersion.Current.Value)
```

The following reference table describes the `MomersionIndicator` constructor:

INDICATORS

MomersionIndicator() 1/3

```
MomersionIndicator QuantConnect.Indicators.MomersionIndicator (
string name,
int? minPeriod,
int fullPeriod
)
```

Initializes a new instance of the `MomersionIndicato` class.

Show Details

Parameters		
`string`	name	The name.
`int?`	minPeriod	The minimum period.
`int`	fullPeriod	The full period.

Return

`MomersionIndicator` - The new `MomersionIndicator` indicator object.

Definition at line 48 of file Indicators/Momersion.cs.

INDICATORS

MomersionIndicator() 2/3

```
MomersionIndicator QuantConnect.Indicators.MomersionIndicator (
int? minPeriod,
int fullPeriod
)
```

Initializes a new instance of the MomersionIndicato class.

Show Details

Parameters		
int?	minPeriod	The minimum period.
int	fullPeriod	The full period.

Return

MomersionIndicator - The new MomersionIndicator indicator object.

Definition at line 65 of file Indicators/Momersion.cs.

INDICATORS

MomersionIndicator() 3/3

```
MomersionIndicator QuantConnect.Indicators.MomersionIndicator (
int fullPeriod
)
```

Initializes a new instance of the MomersionIndicato class.

Show Details

Parameters		
int	fullPeriod	The full period.

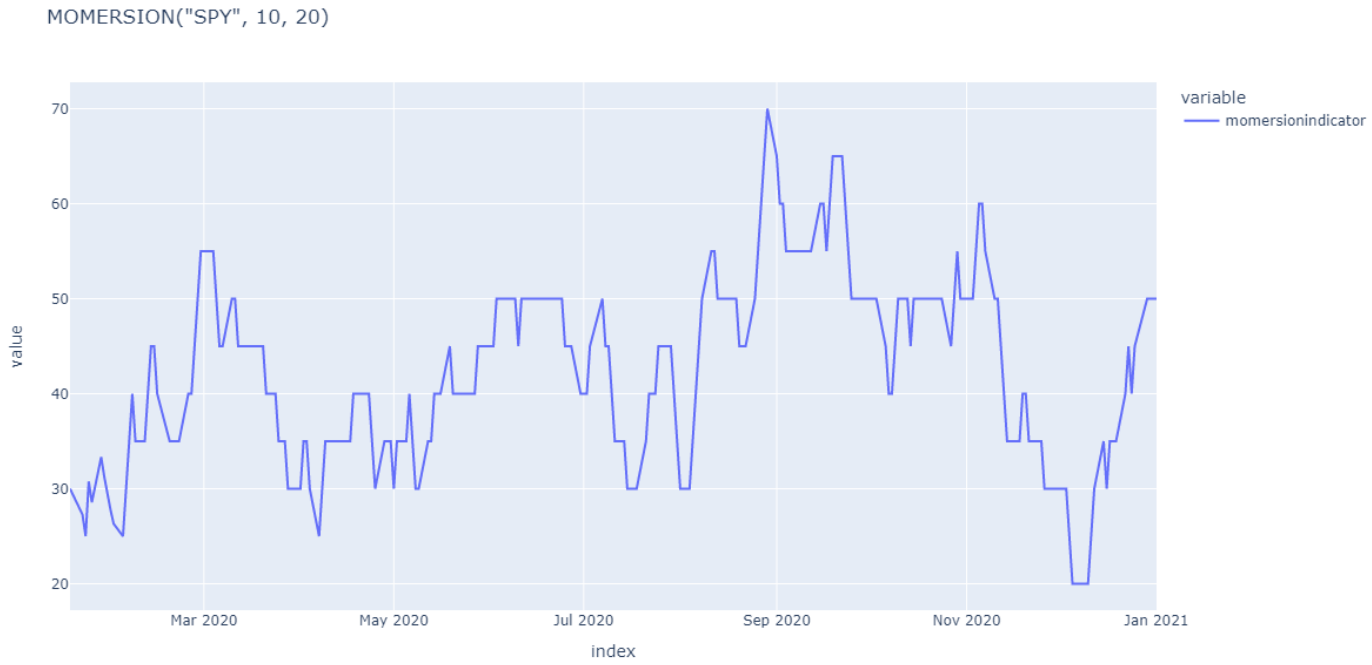
Return

MomersionIndicator - The new MomersionIndicator indicator object.

Definition at line 74 of file Indicators/Momersion.cs.

Visualization

The following image shows plot values of selected properties of `MomersionIndicator` using the plotly library.



Supported Indicators

Money Flow Index

Introduction

The Money Flow Index (MFI) is an oscillator that uses both price and volume to measure buying and selling pressure Typical Price = (High + Low + Close)/3 Money Flow = Typical Price x Volume Positive Money Flow = Sum of the money flows of all days where the typical price is greater than the previous day's typical price Negative Money Flow = Sum of the money flows of all days where the typical price is less than the previous day's typical price Money Flow Ratio = (14-period Positive Money Flow)/(14-period Negative Money Flow) Money Flow Index = 100 x Positive Money Flow / (Positive Money Flow + Negative Money Flow)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MFI Indicator

To create an automatic indicators for `MoneyFlowIndex` , call the `MFI` helper method from the `QCAlgorithm` class. The `MFI` method creates a `MoneyFlowIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class MoneyFlowIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mfi = self.MFI(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.mfi.IsReady:
            # The current value of self.mfi is represented by self.mfi.Current.Value
            self.Plot("MoneyFlowIndex", "mfi", self.mfi.Current.Value)
            # Plot all attributes of self.mfi
            self.Plot("MoneyFlowIndex", "positivemoneyflow", self.mfi.PositiveMoneyFlow.Current.Value)
            self.Plot("MoneyFlowIndex", "negativemoneyflow", self.mfi.NegativeMoneyFlow.Current.Value)
            self.Plot("MoneyFlowIndex", "previoustypicalprice",
self.mfi.PreviousTypicalPrice.Current.Value)
```

The following reference table describes the `MFI` method:

INDICATORS

MFI() 1/1

```
MoneyFlowIndex QuantConnect.Algorithm.QCAlgorithm.MFI (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new MoneyFlowIndex indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MFI we want.
`Int32`	period	The period over which to compute the MFI.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`MoneyFlowIndex` - The MoneyFlowIndex indicator for the requested symbol over the specified period.

Definition at line 1053 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `MoneyFlowIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class MoneyFlowIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mfi = MoneyFlowIndex(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.mfi.Update(bar)

        if self.mfi.IsReady:
            # The current value of self.mfi is represented by self.mfi.Current.Value
            self.Plot("MoneyFlowIndex", "mfi", self.mfi.Current.Value)
            # Plot all attributes of self.mfi
            self.Plot("MoneyFlowIndex", "positivemoneyflow", self.mfi.PositiveMoneyFlow.Current.Value)
            self.Plot("MoneyFlowIndex", "negativemoneyflow", self.mfi.NegativeMoneyFlow.Current.Value)
            self.Plot("MoneyFlowIndex", "previoustypicalprice",
self.mfi.PreviousTypicalPrice.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MoneyFlowIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.mfi = MoneyFlowIndex(20)
        self.RegisterIndicator(self.symbol, self.mfi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.mfi.IsReady:
            # The current value of self.mfi is represented by self.mfi.Current.Value
            self.Plot("MoneyFlowIndex", "mfi", self.mfi.Current.Value)
            # Plot all attributes of self.mfi
            self.Plot("MoneyFlowIndex", "positivemoneyflow", self.mfi.PositiveMoneyFlow.Current.Value)
            self.Plot("MoneyFlowIndex", "negativemoneyflow", self.mfi.NegativeMoneyFlow.Current.Value)
            self.Plot("MoneyFlowIndex", "previoustypicalprice",
self.mfi.PreviousTypicalPrice.Current.Value)
```

The following reference table describes the `MoneyFlowIndex` constructor:

INDICATORS

MoneyFlowIndex() 1/2

```
MoneyFlowIndex QuantConnect.Indicators.MoneyFlowIndex (
int period
)
```

Initializes a new instance of the MoneyFlowIndex class.

Show Details

Parameters		
int	period	The period of the negative and positive money flow.

Return

`MoneyFlowIndex` - The new `MoneyFlowIndex` indicator object.

Definition at line 76 of file Indicators/MoneyFlowIndex.cs.

INDICATORS

MoneyFlowIndex() 2/2

```
MoneyFlowIndex QuantConnect.Indicators.MoneyFlowIndex (
 string name,
 int period
)
```

Initializes a new instance of the MoneyFlowIndex class.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the negative and positive money flow.

Return

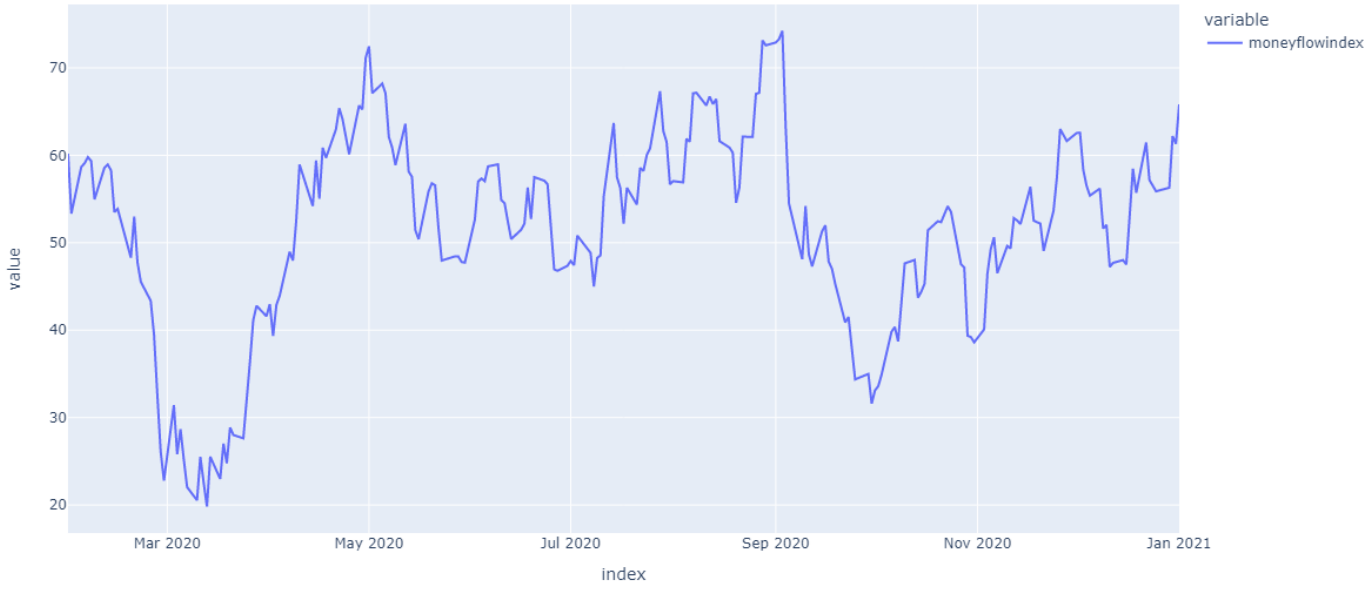
`MoneyFlowIndex` - The new `MoneyFlowIndex` indicator object.

Definition at line 86 of file Indicators/MoneyFlowIndex.cs.

Visualization

The following image shows plot values of selected properties of `MoneyFlowIndex` using the plotly library.

MFI("SPY", 20)


variable
moneyflowindex
70
60
50
40
30
20
value
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Moving Average Convergence Divergence

Introduction

This indicator creates two moving averages defined on a base indicator and produces the difference between the fast and slow averages.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MACD Indicator

To create an automatic indicators for `MovingAverageConvergenceDivergence` , call the `MACD` helper method from the `QCAlgorithm` class. The `MACD` method creates a `MovingAverageConvergenceDivergence` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class MovingAverageConvergenceDivergenceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.macd = self.MACD(self.symbol, 12, 26, 9, MovingAverageType.Exponential)

    def OnData(self, slice: Slice) -> None:
        if self.macd.IsReady:
            # The current value of self.macd is represented by self.macd.Current.Value
            self.Plot("MovingAverageConvergenceDivergence", "macd", self.macd.Current.Value)
            # Plot all attributes of self.macd
            self.Plot("MovingAverageConvergenceDivergence", "fast", self.macd.Fast.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "slow", self.macd.Slow.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "signal", self.macd.Signal.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "histogram",
self.macd.Histogram.Current.Value)
```

The following reference table describes the `MACD` method:

INDICATORS

MACD() 1/1

```
MovingAverageConvergenceDivergence QuantConnect.Algorithm.QCAlgorithm.MACD (
 Symbol symbol,
 Int32 fastPeriod,
 Int32 slowPeriod,
 Int32 signalPeriod,
 *MovingAverageType type,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a MACD indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MACD we want.
`Int32`	fastPeriod	The period for the fast moving average.
`Int32`	slowPeriod	The period for the slow moving average.
`Int32`	signalPeriod	The period for the signal moving average.
`*MovingAverageType`	type	*(Optional)* The type of moving average to use for the MACD.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MovingAverageConvergenceDivergence` - The moving average convergence divergence between the fast and slow averages.

Definition at line 938 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `MovingAverageConvergenceDivergence` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class MovingAverageConvergenceDivergenceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.macd = MovingAverageConvergenceDivergence(12, 26, 9, MovingAverageType.Exponential)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.macd.Update(bar.EndTime, bar.Close)

        if self.macd.IsReady:
            # The current value of self.macd is represented by self.macd.Current.Value
            self.Plot("MovingAverageConvergenceDivergence", "macd", self.macd.Current.Value)
            # Plot all attributes of self.macd
            self.Plot("MovingAverageConvergenceDivergence", "fast", self.macd.Fast.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "slow", self.macd.Slow.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "signal", self.macd.Signal.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "histogram",
self.macd.Histogram.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class MovingAverageConvergenceDivergenceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.macd = MovingAverageConvergenceDivergence(12, 26, 9, MovingAverageType.Exponential)
        self.RegisterIndicator(self.symbol, self.macd, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.macd.IsReady:
            # The current value of self.macd is represented by self.macd.Current.Value
            self.Plot("MovingAverageConvergenceDivergence", "macd", self.macd.Current.Value)
            # Plot all attributes of self.macd
            self.Plot("MovingAverageConvergenceDivergence", "fast", self.macd.Fast.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "slow", self.macd.Slow.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "signal", self.macd.Signal.Current.Value)
            self.Plot("MovingAverageConvergenceDivergence", "histogram",
self.macd.Histogram.Current.Value)
```

The following reference table describes the `MovingAverageConvergenceDivergence` constructor:

INDICATORS

MovingAverageConvergenceDivergence() 1/2

```
MovingAverageConvergenceDivergence QuantConnect.Indicators.MovingAverageConvergenceDivergence (
int fastPeriod,
int slowPeriod,
int signalPeriod,
*MovingAverageType type
)
```

Creates a new MACD with the specified parameters.

Show Details

Parameters		
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
int	signalPeriod	The signal period.
*MovingAverageType	type	*(Optional) (Optional)* The type of moving averages to use. Default: MovingAverageType.Exponential.

Return

MovingAverageConvergenceDivergence - The new MovingAverageConvergenceDivergence indicator object.

Definition at line 64 of file Indicators/MovingAverageConvergenceDivergence.cs.

INDICATORS

MovingAverageConvergenceDivergence() 2/2

```
MovingAverageConvergenceDivergence QuantConnect.Indicators.MovingAverageConvergenceDivergence (
 string name,
 int fastPeriod,
 int slowPeriod,
 int signalPeriod,
 *MovingAverageType type
)
```

Creates a new MACD with the specified parameters.

Show Details

Parameters		
string	name	The name of this indicator.
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
int	signalPeriod	The signal period.
*MovingAverageType	type	*(Optional) (Optional)* The type of moving averages to use. Default: MovingAverageType.Exponential.

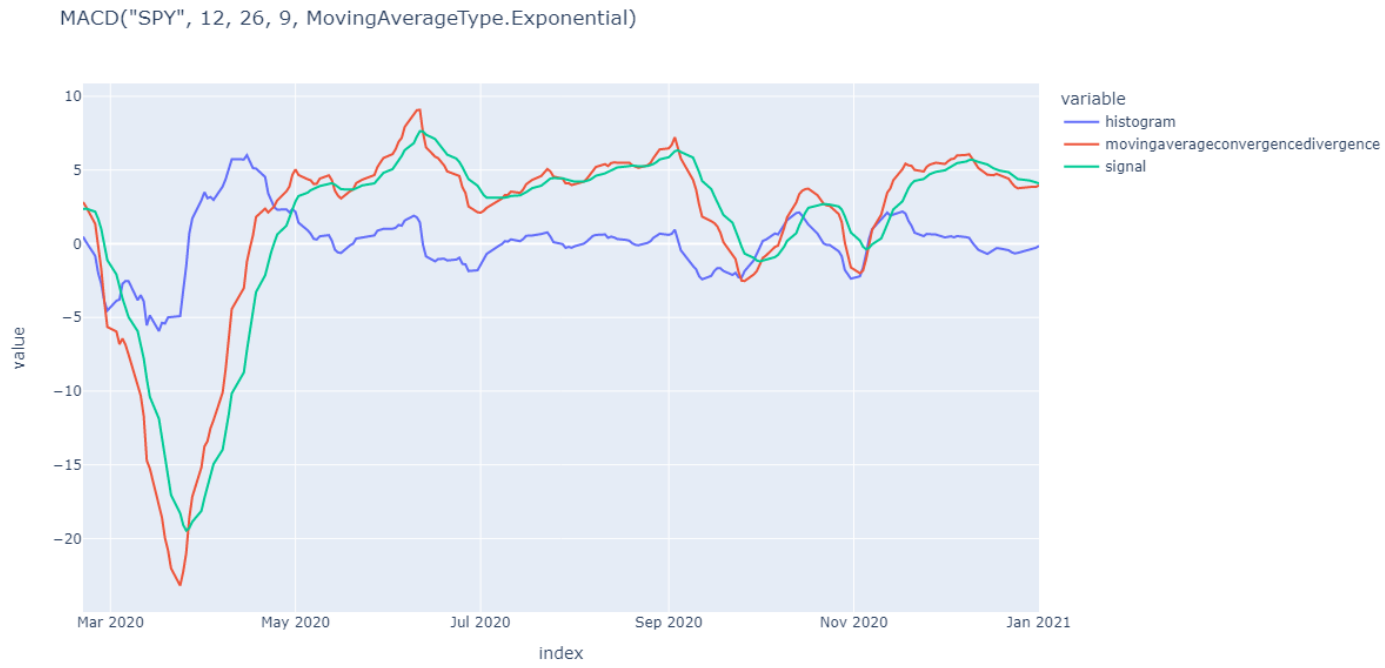
Return

`MovingAverageConvergenceDivergence` - The new `MovingAverageConvergenceDivergence` indicator object.

Definition at line 77 of file Indicators/MovingAverageConvergenceDivergence.cs.

Visualization

The following image shows plot values of selected properties of `MovingAverageConvergenceDivergence` using the plotly library.



Supported Indicators

Normalized Average True Range

Introduction

This indicator computes the Normalized Average True Range (NATR). The Normalized Average True Range is calculated with the following formula: NATR = (ATR(period) / Close) * 100

To view the implementation of this indicator, see the LEAN GitHub repository .

Using NATR Indicator

To create an automatic indicators for `NormalizedAverageTrueRange` , call the `NATR` helper method from the `QCAlgorithm` class. The `NATR` method creates a `NormalizedAverageTrueRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class NormalizedAverageTrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.natr = self.NATR(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.natr.IsReady:
            # The current value of self.natr is represented by self.natr.Current.Value
            self.Plot("NormalizedAverageTrueRange", "natr", self.natr.Current.Value)
```

The following reference table describes the `NATR` method:

INDICATORS

NATR() 1/1

```
NormalizedAverageTrueRange QuantConnect.Algorithm.QCAlgorithm.NATR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new NormalizedAverageTrueRange indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose NATR we want.
`Int32`	period	The period over which to compute the NATR.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`NormalizedAverageTrueRange` - The NormalizedAverageTrueRange indicator for the requested symbol over the specified period.

Definition at line 1220 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `NormalizedAverageTrueRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class NormalizedAverageTrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.natr = NormalizedAverageTrueRange(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.natr.Update(bar)

        if self.natr.IsReady:
            # The current value of self.natr is represented by self.natr.Current.Value
            self.Plot("NormalizedAverageTrueRange", "natr", self.natr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class NormalizedAverageTrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.natr = NormalizedAverageTrueRange(20)
        self.RegisterIndicator(self.symbol, self.natr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.natr.IsReady:
            # The current value of self.natr is represented by self.natr.Current.Value
            self.Plot("NormalizedAverageTrueRange", "natr", self.natr.Current.Value)
```

The following reference table describes the NormalizedAverageTrueRange constructor:

INDICATORS

NormalizedAverageTrueRange() 1/2

```
NormalizedAverageTrueRange QuantConnect.Indicators.NormalizedAverageTrueRange (
string name,
int period
)
```

Initializes a new instance of the NormalizedAverageTrueRang class using the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the NATR.

Return

NormalizedAverageTrueRange - The new NormalizedAverageTrueRange indicator object.

Definition at line 37 of file Indicators/NormalizedAverageTrueRange.cs.

INDICATORS

NormalizedAverageTrueRange() 2/2

```
NormalizedAverageTrueRange QuantConnect.Indicators.NormalizedAverageTrueRange (
int period
)
```

Initializes a new instance of the `NormalizedAverageTrueRang` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the NATR.

Return

`NormalizedAverageTrueRange` - The new `NormalizedAverageTrueRange` indicator object.

Definition at line 49 of file Indicators/NormalizedAverageTrueRange.cs.

Visualization

The following image shows plot values of selected properties of `NormalizedAverageTrueRange` using the plotly library.

Supported Indicators

On Balance Volume

Introduction

This indicator computes the On Balance Volume (OBV). The On Balance Volume is calculated by determining the price of the current close price and previous close price. If the current close price is equivalent to the previous price the OBV remains the same, If the current close price is higher the volume of that day is added to the OBV, while a lower close price will result in negative value.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using OBV Indicator

To create an automatic indicators for `OnBalanceVolume` , call the `OBV` helper method from the `QCAlgorithm` class. The `OBV` method creates a `OnBalanceVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class OnBalanceVolumeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.obv = self.OBV(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.obv.IsReady:
            # The current value of self.obv is represented by self.obv.Current.Value
            self.Plot("OnBalanceVolume", "obv", self.obv.Current.Value)
```

The following reference table describes the `OBV` method:

INDICATORS

OBV() 1/1

```
OnBalanceVolume QuantConnect.Algorithm.QCAlgorithm.OBV (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new On Balance Volume indicator. This will compute the cumulative total volume based on whether the close price being higher or lower than the previous period. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose On Balance Volume we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`OnBalanceVolume` - The On Balance Volume indicator for the requested symbol.

Definition at line 1239 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `OnBalanceVolume` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class OnBalanceVolumeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.obv = OnBalanceVolume()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.obv.Update(bar)

        if self.obv.IsReady:
            # The current value of self.obv is represented by self.obv.Current.Value
            self.Plot("OnBalanceVolume", "obv", self.obv.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class OnBalanceVolumeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.obv = OnBalanceVolume()
        self.RegisterIndicator(self.symbol, self.obv, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.obv.IsReady:
            # The current value of self.obv is represented by self.obv.Current.Value
            self.Plot("OnBalanceVolume", "obv", self.obv.Current.Value)
```

The following reference table describes the `OnBalanceVolume` constructor:

INDICATORS

OnBalanceVolume() 1/2

```
OnBalanceVolume QuantConnect.Indicators.OnBalanceVolume (

)
```

Initializes a new instance of the Indicator class using the specified name.

Show Details

This method requires no argument input.

Return

`OnBalanceVolume` - The new `OnBalanceVolume` indicator object.

Definition at line 34 of file Indicators/OnBalanceVolume.cs.

INDICATORS

OnBalanceVolume() 2/2

```
OnBalanceVolume QuantConnect.Indicators.OnBalanceVolume (
string name
)
```

Initializes a new instance of the Indicator class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

Return

`OnBalanceVolume` - The new `OnBalanceVolume` indicator object.

Definition at line 43 of file Indicators/OnBalanceVolume.cs.

Visualization

The following image shows plot values of selected properties of `OnBalanceVolume` using the plotly library.

Supported Indicators

Parabolic Stop And Reverse

Introduction

Parabolic SAR Indicator Based on TA-Lib implementation

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PSAR Indicator

To create an automatic indicators for `ParabolicStopAndReverse` , call the `PSAR` helper method from the `QCAlgorithm` class. The `PSAR` method creates a `ParabolicStopAndReverse` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class ParabolicStopAndReverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.psar = self.PSAR(self.symbol, 0.02, 0.02, 0.2)

    def OnData(self, slice: Slice) -> None:
        if self.psar.IsReady:
            # The current value of self.psar is represented by self.psar.Current.Value
            self.Plot("ParabolicStopAndReverse", "psar", self.psar.Current.Value)
```

The following reference table describes the `PSAR` method:

INDICATORS

PSAR() 1/1

```
ParabolicStopAndReverse QuantConnect.Algorithm.QCAlgorithm.PSAR (
Symbol symbol,
*Decimal afStart,
*Decimal afIncrement,
*Decimal afMax,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Parabolic SAR indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose PSAR we seek.
`*Decimal`	afStart	*(Optional)* Acceleration factor start value. Normally 0.02.
`*Decimal`	afIncrement	*(Optional)* Acceleration factor increment value. Normally 0.02.
`*Decimal`	afMax	*(Optional)* Acceleration factor max value. Normally 0.2.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`ParabolicStopAndReverse` - A ParabolicStopAndReverse configured with the specified periods.

Definition at line 1299 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `ParabolicStopAndReverse` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class ParabolicStopAndReverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.psar = ParabolicStopAndReverse(0.02, 0.02, 0.2)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.psar.Update(bar)

        if self.psar.IsReady:
            # The current value of self.psar is represented by self.psar.Current.Value
            self.Plot("ParabolicStopAndReverse", "psar", self.psar.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class ParabolicStopAndReverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.psar = ParabolicStopAndReverse(0.02, 0.02, 0.2)
        self.RegisterIndicator(self.symbol, self.psar, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.psar.IsReady:
            # The current value of self.psar is represented by self.psar.Current.Value
            self.Plot("ParabolicStopAndReverse", "psar", self.psar.Current.Value)
```

The following reference table describes the `ParabolicStopAndReverse` constructor:

INDICATORS

ParabolicStopAndReverse() 1/2

```
ParabolicStopAndReverse QuantConnect.Indicators.ParabolicStopAndReverse (
string name,
*decimal afStart,
*decimal afIncrement,
*decimal afMax
)
```

Create new Parabolic SAR.

Show Details

Parameters		
`string`	name	The name of this indicator.
`*decimal`	afStart	*(Optional) (Optional)* Acceleration factor start value. Default: 0.02m.
`*decimal`	afIncrement	*(Optional) (Optional)* Acceleration factor increment value. Default: 0.02m.
`*decimal`	afMax	*(Optional) (Optional)* Acceleration factor max value. Default: 0.2m.

Return

`ParabolicStopAndReverse` - The new `ParabolicStopAndReverse` indicator object.

Definition at line 44 of file Indicators/ParabolicStopAndReverse.cs.

INDICATORS

ParabolicStopAndReverse() 2/2

```
ParabolicStopAndReverse QuantConnect.Indicators.ParabolicStopAndReverse (
*decimal afStart,
*decimal afIncrement,
*decimal afMax
)
```

Create new Parabolic SAR.

Show Details

Parameters		
*decimal	afStart	*(Optional) (Optional)* Acceleration factor start value. Default: 0.02m.
*decimal	afIncrement	*(Optional) (Optional)* Acceleration factor increment value. Default: 0.02m.
*decimal	afMax	*(Optional) (Optional)* Acceleration factor max value. Default: 0.2m.

Return

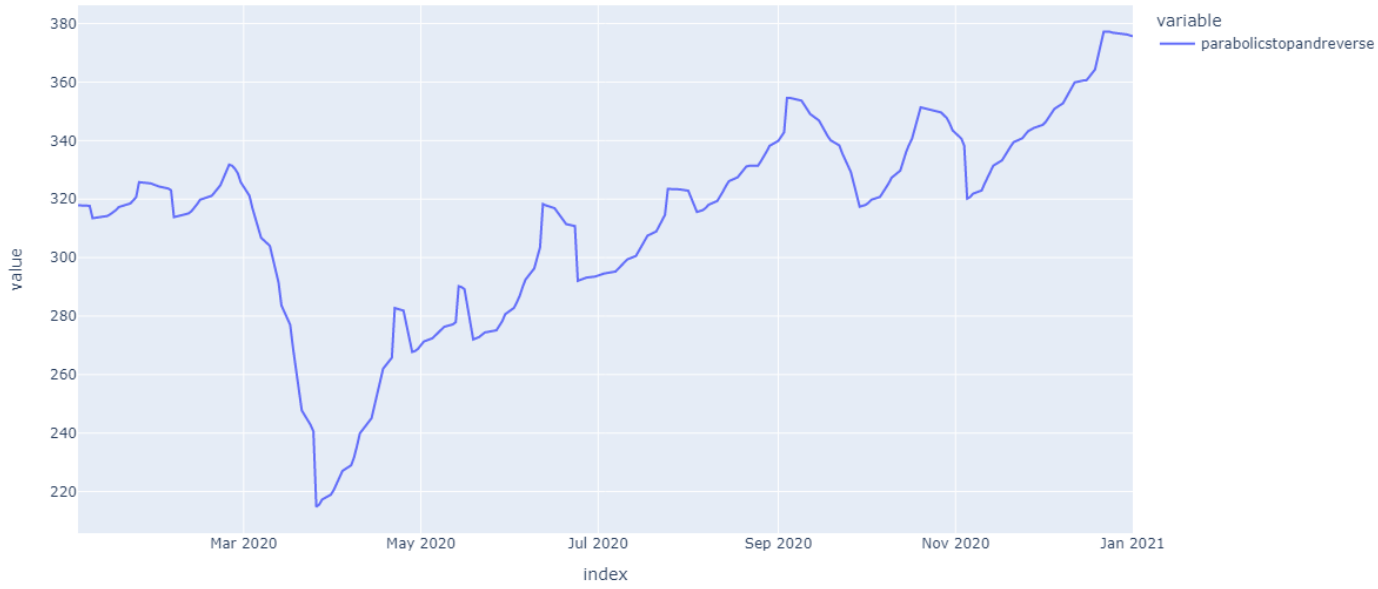
`ParabolicStopAndReverse` - The new `ParabolicStopAndReverse` indicator object.

Definition at line 59 of file Indicators/ParabolicStopAndReverse.cs.

Visualization

The following image shows plot values of selected properties of `ParabolicStopAndReverse` using the plotly library.

PSAR("SPY", 0.02, 0.02, 0.2)



Supported Indicators

Percentage Price Oscillator

Introduction

This indicator computes the Percentage Price Oscillator (PPO) The Percentage Price Oscillator is calculated using the following formula: PPO[i] = 100 * (FastMA[i] - SlowMA[i]) / SlowMA[i]

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PPO Indicator

To create an automatic indicators for `PercentagePriceOscillator` , call the `PPO` helper method from the `QCAlgorithm` class. The `PPO` method creates a `PercentagePriceOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class PercentagePriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ppo = self.PPO(self.symbol, 10, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.ppo.IsReady:
            # The current value of self.ppo is represented by self.ppo.Current.Value
            self.Plot("PercentagePriceOscillator", "ppo", self.ppo.Current.Value)
            # Plot all attributes of self.ppo
            self.Plot("PercentagePriceOscillator", "fast", self.ppo.Fast.Current.Value)
            self.Plot("PercentagePriceOscillator", "slow", self.ppo.Slow.Current.Value)
            self.Plot("PercentagePriceOscillator", "signal", self.ppo.Signal.Current.Value)
            self.Plot("PercentagePriceOscillator", "histogram", self.ppo.Histogram.Current.Value)
```

The following reference table describes the `PPO` method:

INDICATORS

PPO() 1/1

```
PercentagePriceOscillator QuantConnect.Algorithm.QCAlgorithm.PPO (
Symbol symbol,
Int32 fastPeriod,
Int32 slowPeriod,
MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new PercentagePriceOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose PPO we want.
`Int32`	fastPeriod	The fast moving average period.
`Int32`	slowPeriod	The slow moving average period.
`MovingAverageType`	movingAverageType	The type of moving average to use.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`PercentagePriceOscillator` - The PercentagePriceOscillator indicator for the requested symbol over the specified period.

Definition at line 1279 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `PercentagePriceOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class PercentagePriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ppo = PercentagePriceOscillator(10, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ppo.Update(bar.EndTime, bar.Close)

        if self.ppo.IsReady:
            # The current value of self.ppo is represented by self.ppo.Current.Value
            self.Plot("PercentagePriceOscillator", "ppo", self.ppo.Current.Value)
            # Plot all attributes of self.ppo
            self.Plot("PercentagePriceOscillator", "fast", self.ppo.Fast.Current.Value)
            self.Plot("PercentagePriceOscillator", "slow", self.ppo.Slow.Current.Value)
            self.Plot("PercentagePriceOscillator", "signal", self.ppo.Signal.Current.Value)
            self.Plot("PercentagePriceOscillator", "histogram", self.ppo.Histogram.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class PercentagePriceOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ppo = PercentagePriceOscillator(10, 20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.ppo, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ppo.IsReady:
            # The current value of self.ppo is represented by self.ppo.Current.Value
            self.Plot("PercentagePriceOscillator", "ppo", self.ppo.Current.Value)
            # Plot all attributes of self.ppo
            self.Plot("PercentagePriceOscillator", "fast", self.ppo.Fast.Current.Value)
            self.Plot("PercentagePriceOscillator", "slow", self.ppo.Slow.Current.Value)
            self.Plot("PercentagePriceOscillator", "signal", self.ppo.Signal.Current.Value)
            self.Plot("PercentagePriceOscillator", "histogram", self.ppo.Histogram.Current.Value)
```

The following reference table describes the `PercentagePriceOscillator` constructor:

INDICATORS

PercentagePriceOscillator() 1/2

```
PercentagePriceOscillator QuantConnect.Indicators.PercentagePriceOscillator (
string name,
int fastPeriod,
int slowPeriod,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the `PercentagePriceOscillato` class using the specified name and parameters.

Show Details

Parameters		
string	name	The name of this indicator.
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
*MovingAverageType	movingAverageType	*(Optional)* *(Optional)* The type of moving average to use. Default: MovingAverageType.Simple.

Return

PercentagePriceOscillator - The new PercentagePriceOscillator indicator object.

Definition at line 32 of file Indicators/PercentagePriceOscillator.cs.

INDICATORS

PercentagePriceOscillator() 2/2

```
PercentagePriceOscillator QuantConnect.Indicators.PercentagePriceOscillator (
int fastPeriod,
int slowPeriod,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the PercentagePriceOscillato class using the specified parameters.

Show Details

Parameters		
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
*MovingAverageType	movingAverageType	*(Optional)* *(Optional)* The type of moving average to use. Default: MovingAverageType.Simple.

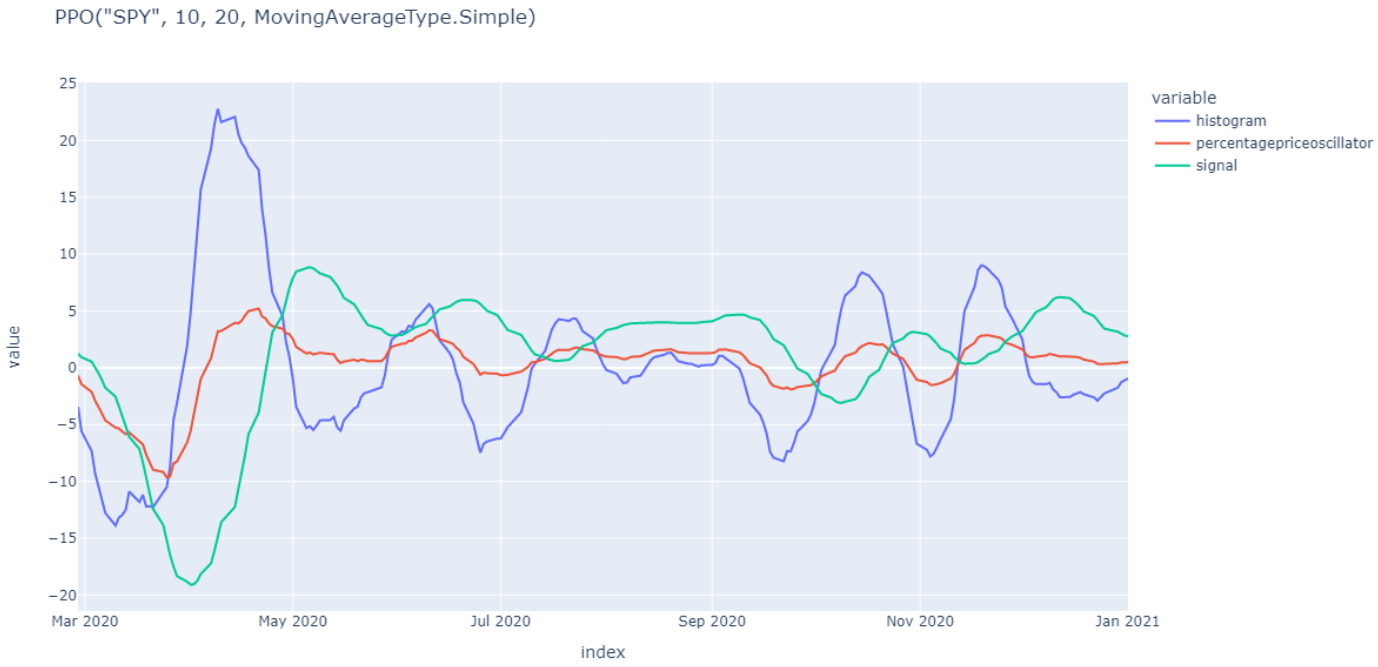
Return

PercentagePriceOscillator - The new PercentagePriceOscillator indicator object.

Definition at line 43 of file Indicators/PercentagePriceOscillator.cs.

Visualization

The following image shows plot values of selected properties of `PercentagePriceOscillator` using the plotly library.



Supported Indicators

Pivot Points High Low

Introduction

Pivot Points (High/Low), also known as Bar Count Reversals, indicator. https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/pivot-points-high-low

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PPHL Indicator

To create an automatic indicators for `PivotPointsHighLow` , call the `PPHL` helper method from the `QCAlgorithm` class. The `PPHL` method creates a `PivotPointsHighLow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class PivotPointsHighLowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.pphl = self.PPHL(self.symbol, 10, 10, 100)

    def OnData(self, slice: Slice) -> None:
        if self.pphl.IsReady:
            # The current value of self.pphl is represented by self.pphl.Current.Value
            self.Plot("PivotPointsHighLow", "pphl", self.pphl.Current.Value)
```

The following reference table describes the `PPHL` method:

INDICATORS

PPHL() 1/1

```
PivotPointsHighLow QuantConnect.Algorithm.QCAlgorithm.PPHL (
Symbol symbol,
Int32 lengthHigh,
Int32 lengthLow,
*Int32 lastStoredValues,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new PivotPointsHighLow indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose PPHL we seek.
`Int32`	lengthHigh	The number of surrounding bars whose high values should be less than the current bar's for the bar high to be marked as high pivot point.
`Int32`	lengthLow	The number of surrounding bars whose low values should be more than the current bar's for the bar low to be marked as low pivot point.
`*Int32`	lastStoredValues	*(Optional)* The number of last stored indicator values.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`PivotPointsHighLow` - The PivotPointsHighLow indicator for the requested symbol.

Definition at line 1259 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `PivotPointsHighLow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class PivotPointsHighLowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.pphl = PivotPointsHighLow(10, 10, 100)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.pphl.Update(bar.EndTime, bar.Close)

        if self.pphl.IsReady:
            # The current value of self.pphl is represented by self.pphl.Current.Value
            self.Plot("PivotPointsHighLow", "pphl", self.pphl.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class PivotPointsHighLowAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.pphl = PivotPointsHighLow(10, 10, 100)
        self.RegisterIndicator(self.symbol, self.pphl, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.pphl.IsReady:
            # The current value of self.pphl is represented by self.pphl.Current.Value
            self.Plot("PivotPointsHighLow", "pphl", self.pphl.Current.Value)
```

The following reference table describes the `PivotPointsHighLow` constructor:

INDICATORS

PivotPointsHighLow() 1/3

```
PivotPointsHighLow QuantConnect.Indicators.PivotPointsHighLow (
int surroundingBarsCount,
*int lastStoredValues
)
```

Creates a new instance of `PivotPointsHighLo` indicator with an equal high and low length.

Show Details

Parameters		
int	surroundingBarsCount	The length parameter here defines the number of surrounding bars that we compare against the current bar high and lows for the max/min.
*int	lastStoredValues	*(Optional) (Optional)* The number of last stored indicator values. Default: 100.

Return

`PivotPointsHighLow` - The new `PivotPointsHighLow` indicator object.

Definition at line 56 of file Indicators/PivotPointsHighLow.cs.

INDICATORS

PivotPointsHighLow() 2/3

```
PivotPointsHighLow QuantConnect.Indicators.PivotPointsHighLow (
 int surroundingBarsCountForHighPoint,
 int surroundingBarsCountForLowPoint,
 *int lastStoredValues
)
```

Creates a new instance of `PivotPointsHighLo` indicator.

Show Details

Parameters		
int	surroundingBarsCountForHighPoint	The number of surrounding bars whose high values should be less than the current bar's for the bar high to be marked as high pivot point.
int	surroundingBarsCountForLowPoint	The number of surrounding bars whose low values should be more than the current bar's for the bar low to be marked as low pivot point.
*int	lastStoredValues	*(Optional) (Optional)* The number of last stored indicator values. Default: 100.

Return

`PivotPointsHighLow` - The new `PivotPointsHighLow` indicator object.

Definition at line 66 of file Indicators/PivotPointsHighLow.cs.

INDICATORS

PivotPointsHighLow() 3/3

```
PivotPointsHighLow QuantConnect.Indicators.PivotPointsHighLow (
 string name,
 int surroundingBarsCountForHighPoint,
 int surroundingBarsCountForLowPoint,
 *int lastStoredValues
 )
```

Creates a new instance of `PivotPointsHighLo` indicator.

Show Details

Parameters		
`string`	name	The name of an indicator.
`int`	surroundingBarsCountForHighPoint	The number of surrounding bars whose high values should be less than the current bar's for the bar high to be marked as high pivot point.
`int`	surroundingBarsCountForLowPoint	The number of surrounding bars whose low values should be more than the current bar's for the bar low to be marked as low pivot point.
`*int`	lastStoredValues	*(Optional) (Optional)* The number of last stored indicator values. Default: 100.

Return

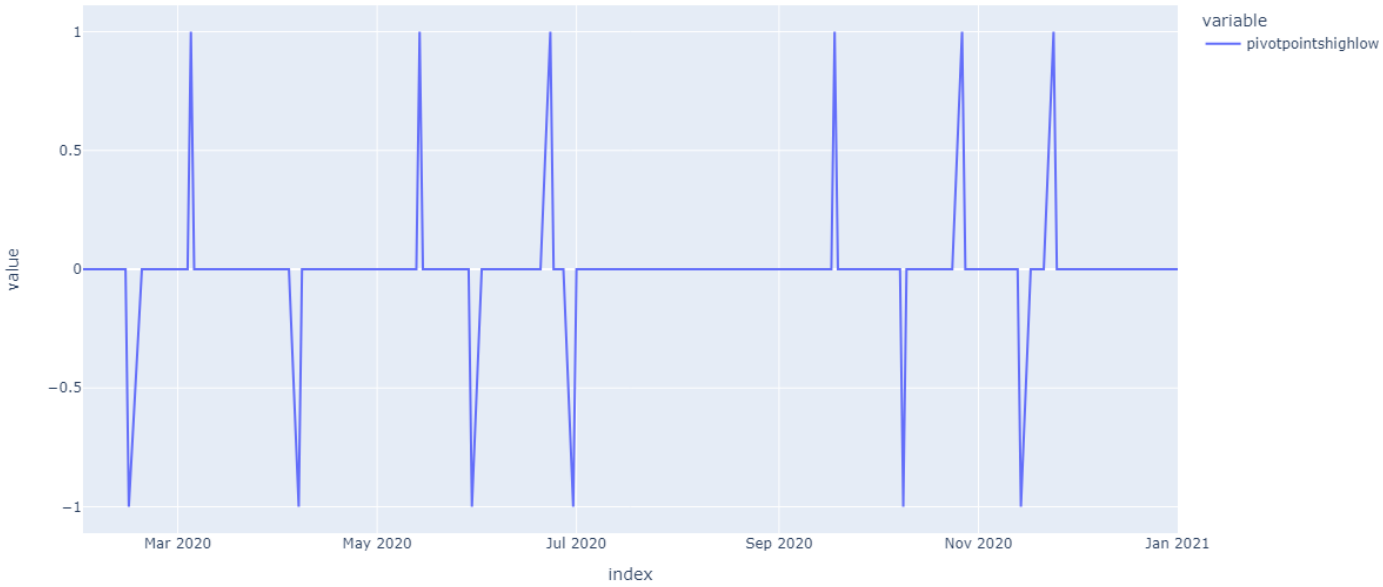
`PivotPointsHighLow` - The new `PivotPointsHighLow` indicator object.

Definition at line 78 of file Indicators/PivotPointsHighLow.cs.

Visualization

The following image shows plot values of selected properties of `PivotPointsHighLow` using the plotly library.

PPHL("SPY", 10, 10, 100)



Supported Indicators

Rate Of Change

Introduction

This indicator computes the n-period rate of change in a value using the following: (value_0 - value_n) / value_n

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ROC Indicator

To create an automatic indicators for `RateOfChange` , call the `ROC` helper method from the `QCAlgorithm` class. The `ROC` method creates a `RateOfChange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class RateOfChangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.roc = self.ROC(self.symbol, 10)

    def OnData(self, slice: Slice) -> None:
        if self.roc.IsReady:
            # The current value of self.roc is represented by self.roc.Current.Value
            self.Plot("RateOfChange", "roc", self.roc.Current.Value)
```

The following reference table describes the `ROC` method:

INDICATORS

ROC() 1/1

```
RateOfChange QuantConnect.Algorithm.QCAlgorithm.ROC (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new RateOfChange indicator. This will compute the n-period rate of change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RateOfChange we want.
`Int32`	period	The period over which to compute the RateOfChange.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RateOfChange` - The RateOfChange indicator for the requested symbol over the specified period.

Definition at line 1356 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RateOfChange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class RateOfChangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.roc = RateOfChange(10)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.roc.Update(bar.EndTime, bar.Close)

        if self.roc.IsReady:
            # The current value of self.roc is represented by self.roc.Current.Value
            self.Plot("RateOfChange", "roc", self.roc.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class RateOfChangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.roc = RateOfChange(10)
        self.RegisterIndicator(self.symbol, self.roc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.roc.IsReady:
            # The current value of self.roc is represented by self.roc.Current.Value
            self.Plot("RateOfChange", "roc", self.roc.Current.Value)
```

The following reference table describes the `RateOfChange` constructor:

INDICATORS

RateOfChange() 1/2

```
RateOfChange QuantConnect.Indicators.RateOfChange (
int period
)
```

Creates a new RateOfChange indicator with the specified period.

Show Details

Parameters		
int	period	The period over which to perform to computation.

Return

`RateOfChange` - The new `RateOfChange` indicator object.

Definition at line 40 of file Indicators/RateOfChange.cs.

INDICATORS

RateOfChange() 2/2

```
RateOfChange QuantConnect.Indicators.RateOfChange (
string name,
int period
)
```

Creates a new RateOfChange indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period over which to perform to computation.

Return

`RateOfChange` - The new `RateOfChange` indicator object.

Definition at line 50 of file Indicators/RateOfChange.cs.

Visualization

The following image shows plot values of selected properties of `RateOfChange` using the plotly library.

Supported Indicators

Rate Of Change Percent

Introduction

This indicator computes the n-period percentage rate of change in a value using the following: 100 * (value_0 - value_n) / value_n

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ROCP Indicator

To create an automatic indicators for `RateOfChangePercent` , call the `ROCP` helper method from the `QCAlgorithm` class. The `ROCP` method creates a `RateOfChangePercent` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class RateOfChangePercentAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rocp = self.ROCP(self.symbol, 10)

    def OnData(self, slice: Slice) -> None:
        if self.rocp.IsReady:
            # The current value of self.rocp is represented by self.rocp.Current.Value
            self.Plot("RateOfChangePercent", "rocp", self.rocp.Current.Value)
```

The following reference table describes the `ROCP` method:

INDICATORS

ROCP() 1/1

```
RateOfChangePercent QuantConnect.Algorithm.QCAlgorithm.ROCP (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RateOfChangePercent indicator. This will compute the n-period percentage rate of change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RateOfChangePercent we want.
`Int32`	period	The period over which to compute the RateOfChangePercent.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RateOfChangePercent` - The RateOfChangePercent indicator for the requested symbol over the specified period.

Definition at line 1375 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RateOfChangePercent` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class RateOfChangePercentAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rocp = RateOfChangePercent(10)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rocp.Update(bar.EndTime, bar.Close)

        if self.rocp.IsReady:
            # The current value of self.rocp is represented by self.rocp.Current.Value
            self.Plot("RateOfChangePercent", "rocp", self.rocp.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class RateOfChangePercentAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rocp = RateOfChangePercent(10)
        self.RegisterIndicator(self.symbol, self.rocp, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rocp.IsReady:
            # The current value of self.rocp is represented by self.rocp.Current.Value
            self.Plot("RateOfChangePercent", "rocp", self.rocp.Current.Value)
```

The following reference table describes the `RateOfChangePercent` constructor:

INDICATORS

RateOfChangePercent() 1/2

```
RateOfChangePercent QuantConnect.Indicators.RateOfChangePercent (
int period
)
```

Creates a new RateOfChangePercent indicator with the specified period.

Show Details

Parameters		
int	period	The period over which to perform to computation.

Return

`RateOfChangePercent` - The new `RateOfChangePercent` indicator object.

Definition at line 28 of file Indicators/RateOfChangePercent.cs.

INDICATORS

RateOfChangePercent() 2/2

```
RateOfChangePercent QuantConnect.Indicators.RateOfChangePercent (
string name,
int period
)
```

Creates a new RateOfChangePercent indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period over which to perform to computation.

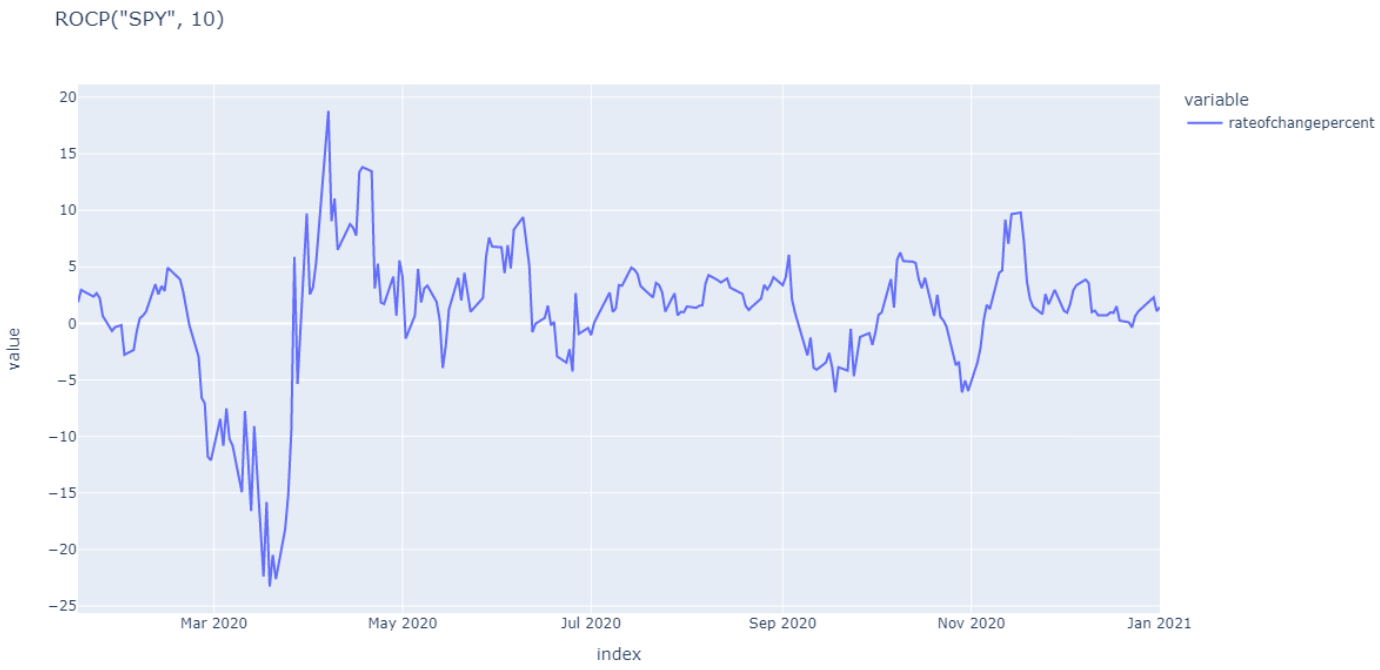
Return

`RateOfChangePercent` - The new `RateOfChangePercent` indicator object.

Definition at line 38 of file Indicators/RateOfChangePercent.cs.

Visualization

The following image shows plot values of selected properties of `RateOfChangePercent` using the plotly library.



Supported Indicators

Rate Of Change Ratio

Introduction

This indicator computes the Rate Of Change Ratio (ROCR). The Rate Of Change Ratio is calculated with the following formula: ROCR = price / prevPrice

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ROCR Indicator

To create an automatic indicators for `RateOfChangeRatio` , call the `ROCR` helper method from the `QCAlgorithm` class. The `ROCR` method creates a `RateOfChangeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class RateOfChangeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rocr = self.ROCR(self.symbol, 10)

    def OnData(self, slice: Slice) -> None:
        if self.rocr.IsReady:
            # The current value of self.rocr is represented by self.rocr.Current.Value
            self.Plot("RateOfChangeRatio", "rocr", self.rocr.Current.Value)
```

The following reference table describes the `ROCR` method:

INDICATORS

ROCR() 1/1

```
RateOfChangeRatio QuantConnect.Algorithm.QCAlgorithm.ROCR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RateOfChangeRatio indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ROCR we want.
`Int32`	period	The period over which to compute the ROCR.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RateOfChangeRatio` - The RateOfChangeRatio indicator for the requested symbol over the specified period.

Definition at line 1393 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RateOfChangeRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class RateOfChangeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rocr = RateOfChangeRatio(10)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rocr.Update(bar.EndTime, bar.Close)

        if self.rocr.IsReady:
            # The current value of self.rocr is represented by self.rocr.Current.Value
            self.Plot("RateOfChangeRatio", "rocr", self.rocr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class RateOfChangeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rocr = RateOfChangeRatio(10)
        self.RegisterIndicator(self.symbol, self.rocr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rocr.IsReady:
            # The current value of self.rocr is represented by self.rocr.Current.Value
            self.Plot("RateOfChangeRatio", "rocr", self.rocr.Current.Value)
```

The following reference table describes the `RateOfChangeRatio` constructor:

INDICATORS

RateOfChangeRatio() 1/2

```
RateOfChangeRatio QuantConnect.Indicators.RateOfChangeRatio (
string name,
int period
)
```

Initializes a new instance of the `RateOfChangeRati` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the ROCR.

Return

`RateOfChangeRatio` - The new `RateOfChangeRatio` indicator object.

Definition at line 30 of file Indicators/RateOfChangeRatio.cs.

INDICATORS

RateOfChangeRatio() 2/2

```
RateOfChangeRatio QuantConnect.Indicators.RateOfChangeRatio (
int period
)
```

Initializes a new instance of the `RateOfChangeRati` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the ROCR.

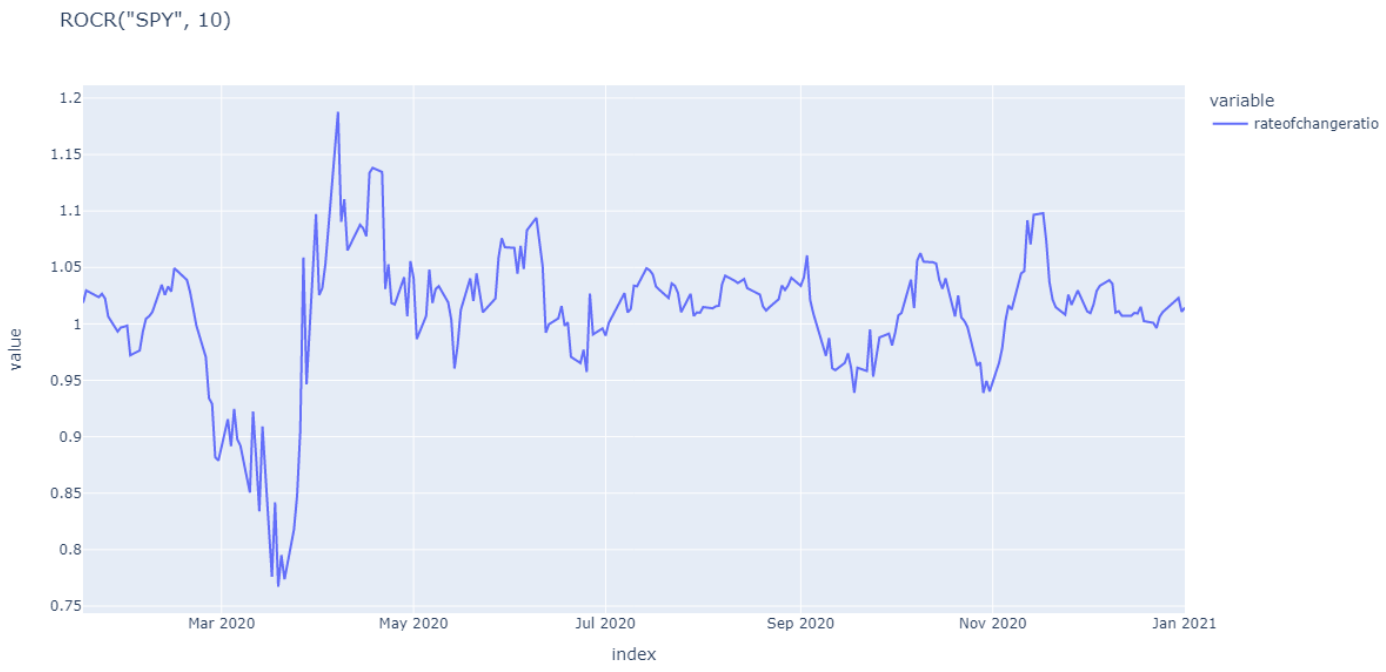
Return

`RateOfChangeRatio` - The new `RateOfChangeRatio` indicator object.

Definition at line 39 of file Indicators/RateOfChangeRatio.cs.

Visualization

The following image shows plot values of selected properties of `RateOfChangeRatio` using the plotly library.



Supported Indicators

Regression Channel

Introduction

The Regression Channel indicator extends the with the inclusion of two (upper and lower) channel lines that are distanced from the linear regression line by a user defined number of standard deviations. Reference: http://www.onlinetradingconcepts.com/TechnicalAnalysis/LinRegChannel.html

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RC Indicator

To create an automatic indicators for `RegressionChannel` , call the `RC` helper method from the `QCAlgorithm` class. The `RC` method creates a `RegressionChannel` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class RegressionChannelAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rc = self.RC(self.symbol, 20, 2)

    def OnData(self, slice: Slice) -> None:
        if self.rc.IsReady:
            # The current value of self.rc is represented by self.rc.Current.Value
            self.Plot("RegressionChannel", "rc", self.rc.Current.Value)
            # Plot all attributes of self.rc
            self.Plot("RegressionChannel", "linearregression", self.rc.LinearRegression.Current.Value)
            self.Plot("RegressionChannel", "upperchannel", self.rc.UpperChannel.Current.Value)
            self.Plot("RegressionChannel", "lowerchannel", self.rc.LowerChannel.Current.Value)
            self.Plot("RegressionChannel", "intercept", self.rc.Intercept.Current.Value)
            self.Plot("RegressionChannel", "slope", self.rc.Slope.Current.Value)
```

The following reference table describes the `RC` method:

INDICATORS

RC() 1/1

```
RegressionChannel QuantConnect.Algorithm.QCAlgorithm.RC (
Symbol symbol,
Int32 period,
Decimal k,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RegressionChannel indicator which will compute the LinearRegression, UpperChannel and

LowerChannel lines, the intercept and slope.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RegressionChannel we seek.
`Int32`	period	The period of the standard deviation and least square moving average (linear regression line).
`Decimal`	k	The number of standard deviations specifying the distance between the linear regression and upper or lower channel lines.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RegressionChannel` - A Regression Channel configured with the specified period and number of standard deviation.

Definition at line 1318 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RegressionChannel` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```
class RegressionChannelAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rc = RegressionChannel(20, 2)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rc.Update(bar.EndTime, bar.Close)

        if self.rc.IsReady:
            # The current value of self.rc is represented by self.rc.Current.Value
            self.Plot("RegressionChannel", "rc", self.rc.Current.Value)
            # Plot all attributes of self.rc
            self.Plot("RegressionChannel", "linearregression", self.rc.LinearRegression.Current.Value)
            self.Plot("RegressionChannel", "upperchannel", self.rc.UpperChannel.Current.Value)
            self.Plot("RegressionChannel", "lowerchannel", self.rc.LowerChannel.Current.Value)
            self.Plot("RegressionChannel", "intercept", self.rc.Intercept.Current.Value)
            self.Plot("RegressionChannel", "slope", self.rc.Slope.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class RegressionChannelAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rc = RegressionChannel(20, 2)
        self.RegisterIndicator(self.symbol, self.rc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rc.IsReady:
            # The current value of self.rc is represented by self.rc.Current.Value
            self.Plot("RegressionChannel", "rc", self.rc.Current.Value)
            # Plot all attributes of self.rc
            self.Plot("RegressionChannel", "linearregression", self.rc.LinearRegression.Current.Value)
            self.Plot("RegressionChannel", "upperchannel", self.rc.UpperChannel.Current.Value)
            self.Plot("RegressionChannel", "lowerchannel", self.rc.LowerChannel.Current.Value)
            self.Plot("RegressionChannel", "intercept", self.rc.Intercept.Current.Value)
            self.Plot("RegressionChannel", "slope", self.rc.Slope.Current.Value)
```

The following reference table describes the `RegressionChannel` constructor:

INDICATORS

RegressionChannel() 1/2

```
RegressionChannel QuantConnect.Indicators.RegressionChannel (
string name,
int period,
decimal k
)
```

Initializes a new instance of the `RegressionChanne` class.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The number of data points to hold in the window.
decimal	k	The number of standard deviations specifying the distance between the linear regression and upper or lower channel lines.

Return

`RegressionChannel` - The new `RegressionChannel` indicator object.

Definition at line 72 of file Indicators/RegressionChannel.cs.

INDICATORS

RegressionChannel() 2/2

```
RegressionChannel QuantConnect.Indicators.RegressionChannel (
int period,
decimal k
)
```

Initializes a new instance of the `LeastSquaresMovingAverag` class.

Show Details

Parameters		
int	period	The number of data points to hold in the window.
decimal	k	The number of standard deviations specifying the distance between the linear regression and upper or lower channel lines.

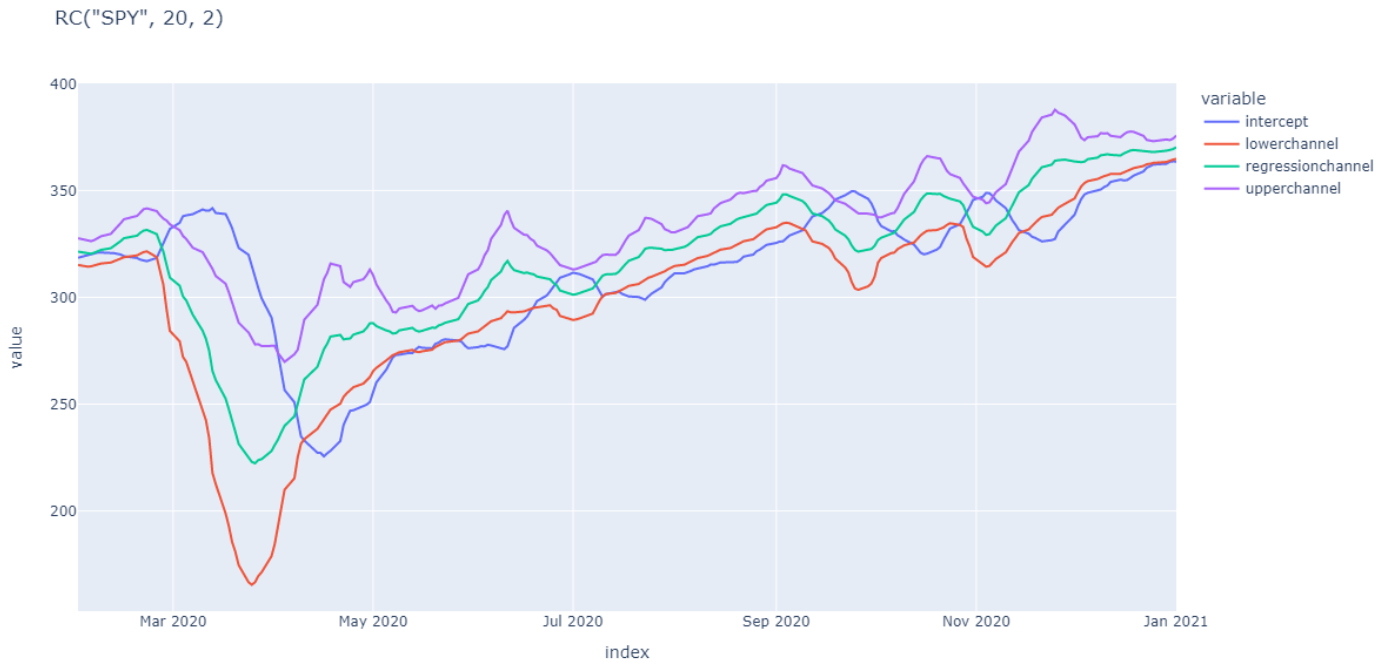
Return

`RegressionChannel` - The new `RegressionChannel` indicator object.

Definition at line 87 of file Indicators/RegressionChannel.cs.

Visualization

The following image shows plot values of selected properties of `RegressionChannel` using the plotly library.



Supported Indicators

Relative Daily Volume

Introduction

The Relative Daily Volume indicator is an indicator that compares current cumulative volume to the cumulative volume for a given time of day, measured as a ratio. Current volume from open to current time of day / Average over the past x days from open to current time of day

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RDV Indicator

To create an automatic indicators for `RelativeDailyVolume` , call the `RDV` helper method from the `QCAlgorithm` class. The `RDV` method creates a `RelativeDailyVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class RelativeDailyVolumeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rdv = self.RDV(self.symbol, 2)

    def OnData(self, slice: Slice) -> None:
        if self.rdv.IsReady:
            # The current value of self.rdv is represented by self.rdv.Current.Value
            self.Plot("RelativeDailyVolume", "rdv", self.rdv.Current.Value)
```

The following reference table describes the `RDV` method:

INDICATORS

RDV() 1/1

```
RelativeDailyVolume QuantConnect.Algorithm.QCAlgorithm.RDV (
Symbol symbol,
*Int32 period,
*Resolution resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an RelativeDailyVolume indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RDV we want.
`*Int32`	period	*(Optional)* The period of the RDV.
`*Resolution`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`RelativeDailyVolume` - The Relative Volume indicator for the given parameters.

Definition at line 1450 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RelativeDailyVolume` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class RelativeDailyVolumeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rdv = RelativeDailyVolume(2)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rdv.Update(bar)

        if self.rdv.IsReady:
            # The current value of self.rdv is represented by self.rdv.Current.Value
            self.Plot("RelativeDailyVolume", "rdv", self.rdv.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class RelativeDailyVolumeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rdv = RelativeDailyVolume(2)
        self.RegisterIndicator(self.symbol, self.rdv, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rdv.IsReady:
            # The current value of self.rdv is represented by self.rdv.Current.Value
            self.Plot("RelativeDailyVolume", "rdv", self.rdv.Current.Value)
```

The following reference table describes the `RelativeDailyVolume` constructor:

INDICATORS

RelativeDailyVolume() 1/2

```
RelativeDailyVolume QuantConnect.Indicators.RelativeDailyVolume (
*int period
)
```

Initializes a new instance of the RelativeDailyVolume class using the specified period.

Show Details

Parameters		
*int	period	*(Optional) (Optional)* The period over which to perform the computation. Default: 2.

Return

`RelativeDailyVolume` - The new `RelativeDailyVolume` indicator object.

Definition at line 50 of file Indicators/RelativeDailyVolume.cs.

INDICATORS

RelativeDailyVolume() 2/2

```
RelativeDailyVolume QuantConnect.Indicators.RelativeDailyVolume (
string name,
int period
)
```

Creates a new RelativeDailyVolume indicator with the specified period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of this indicator.

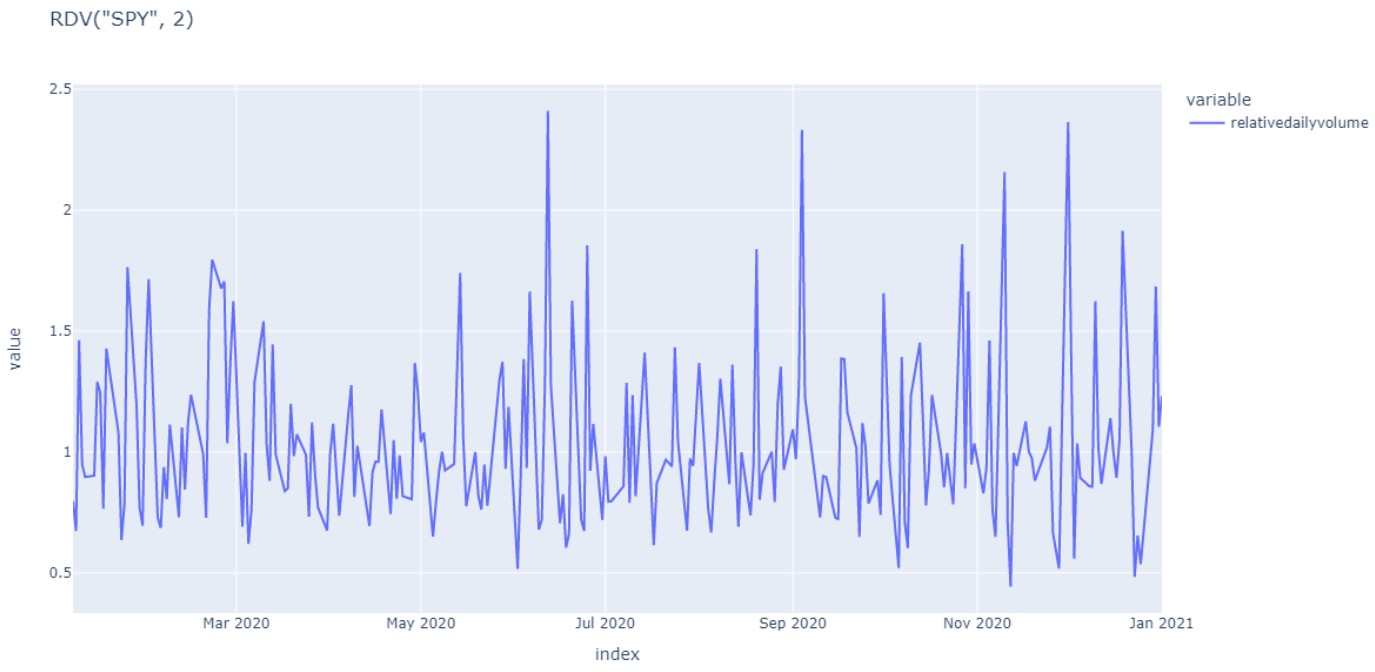
Return

`RelativeDailyVolume` - The new `RelativeDailyVolume` indicator object.

Definition at line 60 of file Indicators/RelativeDailyVolume.cs.

Visualization

The following image shows plot values of selected properties of `RelativeDailyVolume` using the plotly library.



Supported Indicators

Relative Moving Average

Introduction

This indicator represents the relative moving average indicator (RMA). RMA = SMA(3 x Period) - SMA(2 x Period) + SMA(1 x Period) per formula: https://www.hybrid-solutions.com/plugins/client-vtl-plugins/free/rma.html

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RMA Indicator

To create an automatic indicators for `RelativeMovingAverage` , call the `RMA` helper method from the `QCAlgorithm` class. The `RMA` method creates a `RelativeMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class RelativeMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rma = self.RMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.rma.IsReady:
            # The current value of self.rma is represented by self.rma.Current.Value
            self.Plot("RelativeMovingAverage", "rma", self.rma.Current.Value)
            # Plot all attributes of self.rma
            self.Plot("RelativeMovingAverage", "shortaverage", self.rma.ShortAverage.Current.Value)
            self.Plot("RelativeMovingAverage", "mediumaverage", self.rma.MediumAverage.Current.Value)
            self.Plot("RelativeMovingAverage", "longaverage", self.rma.LongAverage.Current.Value)
```

The following reference table describes the `RMA` method:

INDICATORS

RMA() 1/1

```
RelativeMovingAverage QuantConnect.Algorithm.QCAlgorithm.RMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Relative Moving Average indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose relative moving average we seek.
`Int32`	period	The period of the relative moving average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RelativeMovingAverage` - A relative moving average configured with the specified period and number of standard deviation.

Definition at line 1336 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `RelativeMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class RelativeMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rma = RelativeMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rma.Update(bar.EndTime, bar.Close)

        if self.rma.IsReady:
            # The current value of self.rma is represented by self.rma.Current.Value
            self.Plot("RelativeMovingAverage", "rma", self.rma.Current.Value)
            # Plot all attributes of self.rma
            self.Plot("RelativeMovingAverage", "shortaverage", self.rma.ShortAverage.Current.Value)
            self.Plot("RelativeMovingAverage", "mediumaverage", self.rma.MediumAverage.Current.Value)
            self.Plot("RelativeMovingAverage", "longaverage", self.rma.LongAverage.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class RelativeMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rma = RelativeMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.rma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rma.IsReady:
            # The current value of self.rma is represented by self.rma.Current.Value
            self.Plot("RelativeMovingAverage", "rma", self.rma.Current.Value)
            # Plot all attributes of self.rma
            self.Plot("RelativeMovingAverage", "shortaverage", self.rma.ShortAverage.Current.Value)
            self.Plot("RelativeMovingAverage", "mediumaverage", self.rma.MediumAverage.Current.Value)
            self.Plot("RelativeMovingAverage", "longaverage", self.rma.LongAverage.Current.Value)
```

The following reference table describes the `RelativeMovingAverage` constructor:

INDICATORS

RelativeMovingAverage() 1/2

```
RelativeMovingAverage QuantConnect.Indicators.RelativeMovingAverage (
string name,
int period
)
```

Initializes a new instance of the RelativeMovingAverage class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the RMA.

Return

`RelativeMovingAverage` - The new `RelativeMovingAverage` indicator object.

Definition at line 55 of file Indicators/RelativeMovingAverage.cs.

INDICATORS

RelativeMovingAverage() 2/2

```
RelativeMovingAverage QuantConnect.Indicators.RelativeMovingAverage (
int period
)
```

Initializes a new instance of the SimpleMovingAverage class with the default name and period.

Show Details

Parameters		
int	period	/

Return

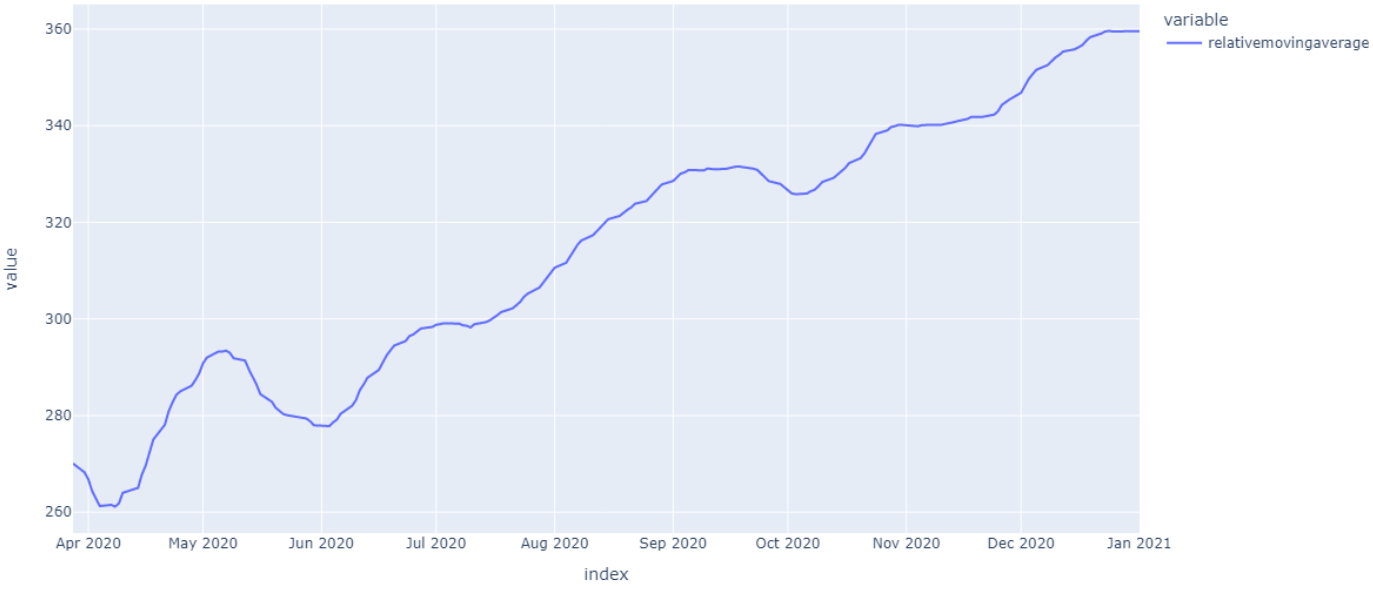
`RelativeMovingAverage` - The new `RelativeMovingAverage` indicator object.

Definition at line 67 of file Indicators/RelativeMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `RelativeMovingAverage` using the plotly library.

RMA("SPY", 20)


variable
relativemovingaverage
360
340
320
300
280
260
value
Apr 2020
May 2020
Jun 2020
Jul 2020
Aug 2020
Sep 2020
Oct 2020
Nov 2020
Dec 2020
Jan 2021
index

Supported Indicators

Relative Strength Index

Introduction

This indicator represents the Relative Strength Index (RSI) developed by K. Welles Wilder. You can optionally specified a different moving average type to be used in the computation

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RSI Indicator

To create an automatic indicators for `RelativeStrengthIndex` , call the `RSI` helper method from the `QCAlgorithm` class. The `RSI` method creates a `RelativeStrengthIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class RelativeStrengthIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rsi = self.RSI(self.symbol, 14)

    def OnData(self, slice: Slice) -> None:
        if self.rsi.IsReady:
            # The current value of self.rsi is represented by self.rsi.Current.Value
            self.Plot("RelativeStrengthIndex", "rsi", self.rsi.Current.Value)
            # Plot all attributes of self.rsi
            self.Plot("RelativeStrengthIndex", "averageloss", self.rsi.AverageLoss.Current.Value)
            self.Plot("RelativeStrengthIndex", "averagegain", self.rsi.AverageGain.Current.Value)
```

The following reference table describes the `RSI` method:

INDICATORS

RSI() 1/1

```
RelativeStrengthIndex QuantConnect.Algorithm.QCAlgorithm.RSI (
Symbol symbol,
Int32 period,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RelativeStrengthIndex indicator. This will produce an oscillator that ranges from 0 to 100 based on the ratio of average gains to average losses over the specified period.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RSI we want.
`Int32`	period	The period over which to compute the RSI.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use in computing the average gain/loss values.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RelativeStrengthIndex` - The RelativeStrengthIndex indicator for the requested symbol over the specified period.

Definition at line 1413 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `RelativeStrengthIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class RelativeStrengthIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rsi = RelativeStrengthIndex(14)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rsi.Update(bar.EndTime, bar.Close)

        if self.rsi.IsReady:
            # The current value of self.rsi is represented by self.rsi.Current.Value
            self.Plot("RelativeStrengthIndex", "rsi", self.rsi.Current.Value)
            # Plot all attributes of self.rsi
            self.Plot("RelativeStrengthIndex", "averageloss", self.rsi.AverageLoss.Current.Value)
            self.Plot("RelativeStrengthIndex", "averagegain", self.rsi.AverageGain.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class RelativeStrengthIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rsi = RelativeStrengthIndex(14)
        self.RegisterIndicator(self.symbol, self.rsi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rsi.IsReady:
            # The current value of self.rsi is represented by self.rsi.Current.Value
            self.Plot("RelativeStrengthIndex", "rsi", self.rsi.Current.Value)
            # Plot all attributes of self.rsi
            self.Plot("RelativeStrengthIndex", "averageloss", self.rsi.AverageLoss.Current.Value)
            self.Plot("RelativeStrengthIndex", "averagegain", self.rsi.AverageGain.Current.Value)
```

The following reference table describes the `RelativeStrengthIndex` constructor:

INDICATORS

RelativeStrengthIndex() 1/2

```
RelativeStrengthIndex QuantConnect.Indicators.RelativeStrengthIndex (
int period,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the RelativeStrengthIndex class with the specified name and period.

Show Details

Parameters		
int	period	The period used for up and down days.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used for computing the average gain/loss values. Default: MovingAverageType.Wilders.

Return

RelativeStrengthIndex - The new RelativeStrengthIndex indicator object.

Definition at line 48 of file Indicators/RelativeStrengthIndex.cs.

INDICATORS

RelativeStrengthIndex() 2/2

```
RelativeStrengthIndex QuantConnect.Indicators.RelativeStrengthIndex (
string name,
int period,
*MovingAverageType movingAverageType
)
```

Initializes a new instance of the RelativeStrengthIndex class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period used for up and down days.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of moving average to be used for computing the average gain/loss values. Default: MovingAverageType.Wilders.

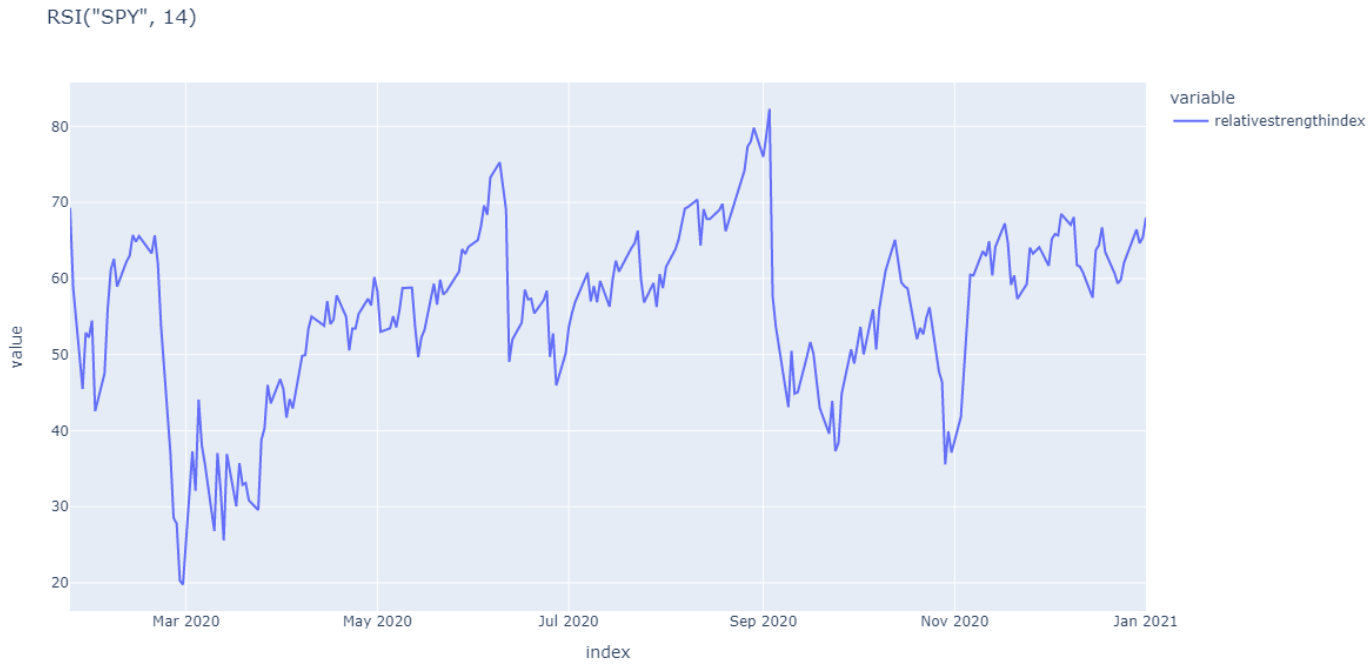
Return

RelativeStrengthIndex - The new RelativeStrengthIndex indicator object.

Definition at line 59 of file Indicators/RelativeStrengthIndex.cs.

Visualization

The following image shows plot values of selected properties of `RelativeStrengthIndex` using the plotly library.



Supported Indicators

Relative Vigor Index

Introduction

The Relative Vigor Index (RVI) compares the ratio of the closing price of a security to its trading range. For illustration, let: a = Close−Open b = Close−Open of One Bar Prior to a c = Close−Open of One Bar Prior to b d = Close−Open of One Bar Prior to c e = High−Low of Bar a f = High−Low of Bar b g = High−Low of Bar c h = High−Low of Bar d Then let (a+2*(b+c)+d)/6 be NUM and (e+2*(f+g)+h)/6 be DENOM. RVI = SMA(NUM)/SMA(DENOM) for a specified period. https://www.investopedia.com/terms/r/relative_vigor_index.asp

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RVI Indicator

To create an automatic indicators for `RelativeVigorIndex` , call the `RVI` helper method from the `QCAlgorithm` class. The `RVI` method creates a `RelativeVigorIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class RelativeVigorIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rvi = self.RVI(self.symbol, 20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        if self.rvi.IsReady:
            # The current value of self.rvi is represented by self.rvi.Current.Value
            self.Plot("RelativeVigorIndex", "rvi", self.rvi.Current.Value)
            # Plot all attributes of self.rvi
            self.Plot("RelativeVigorIndex", "signal", self.rvi.Signal.Current.Value)
```

The following reference table describes the `RVI` method:

INDICATORS

RVI() 1/1

```
RelativeVigorIndex QuantConnect.Algorithm.QCAlgorithm.RVI (
 Symbol symbol,
 Int32 period,
 *MovingAverageType movingAverageType,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new RelativeVigorIndex indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RVI we want.
`Int32`	period	The period over which to compute the RVI.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`RelativeVigorIndex` - The RelativeVigorIndex indicator for the requested symbol over the specified period.

Definition at line 1431 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `RelativeVigorIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class RelativeVigorIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rvi = RelativeVigorIndex(20, MovingAverageType.Simple)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.rvi.Update(bar)

        if self.rvi.IsReady:
            # The current value of self.rvi is represented by self.rvi.Current.Value
            self.Plot("RelativeVigorIndex", "rvi", self.rvi.Current.Value)
            # Plot all attributes of self.rvi
            self.Plot("RelativeVigorIndex", "signal", self.rvi.Signal.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class RelativeVigorIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.rvi = RelativeVigorIndex(20, MovingAverageType.Simple)
        self.RegisterIndicator(self.symbol, self.rvi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.rvi.IsReady:
            # The current value of self.rvi is represented by self.rvi.Current.Value
            self.Plot("RelativeVigorIndex", "rvi", self.rvi.Current.Value)
            # Plot all attributes of self.rvi
            self.Plot("RelativeVigorIndex", "signal", self.rvi.Signal.Current.Value)
```

The following reference table describes the `RelativeVigorIndex` constructor:

INDICATORS

RelativeVigorIndex() 1/2

```
RelativeVigorIndex QuantConnect.Indicators.RelativeVigorIndex (
int period,
MovingAverageType type
)
```

Initializes a new instance of the `RelativeVigorInde` (RVI) class.

Show Details

Parameters		
`int`	period	The period for the RelativeVigorIndex.
`MovingAverageType`	type	The type of Moving Average to use.

Return

`RelativeVigorIndex` - The new `RelativeVigorIndex` indicator object.

Definition at line 72 of file Indicators/RelativeVigorIndex.cs.

INDICATORS

RelativeVigorIndex() 2/2

```
RelativeVigorIndex QuantConnect.Indicators.RelativeVigorIndex (
string name,
int period,
*MovingAverageType type
)
```

Initializes a new instance of the `RelativeVigorInde` (RVI) class.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period for the RelativeVigorIndex.
`*MovingAverageType`	type	*(Optional) (Optional)* The type of Moving Average to use. Default: MovingAverageType.Simple.

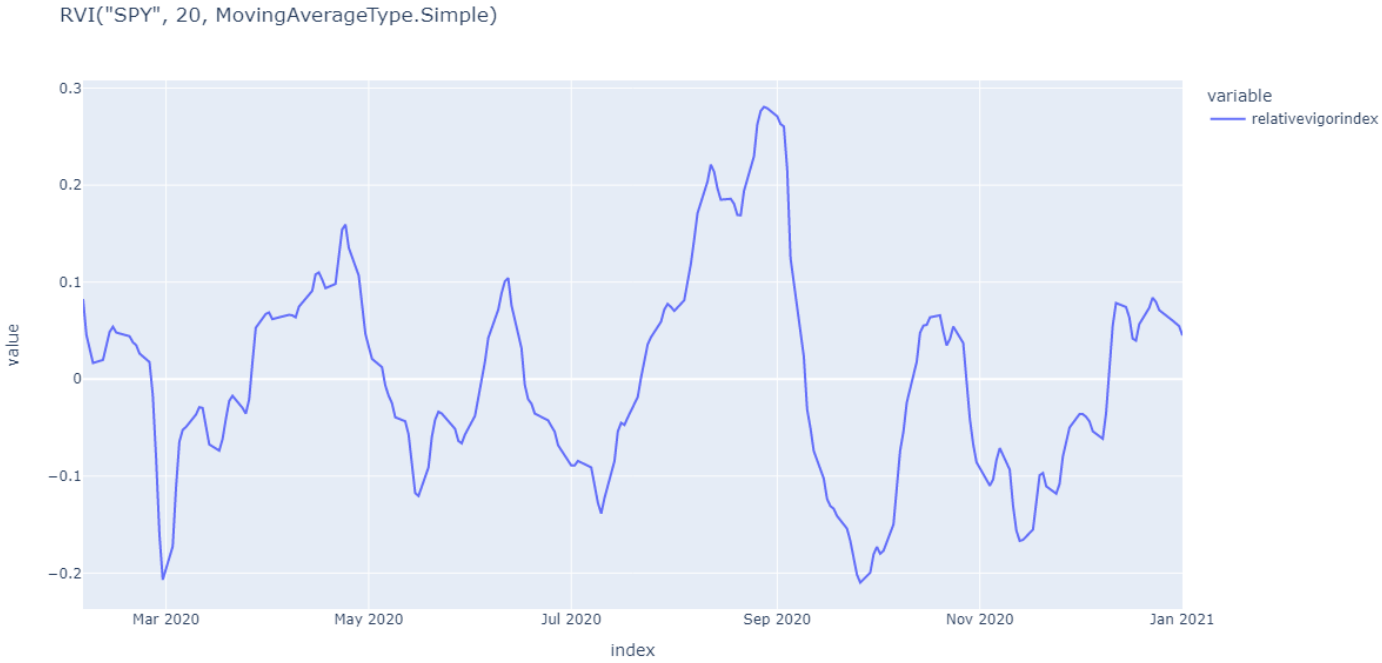
Return

`RelativeVigorIndex` - The new `RelativeVigorIndex` indicator object.

Definition at line 83 of file Indicators/RelativeVigorIndex.cs.

Visualization

The following image shows plot values of selected properties of `RelativeVigorIndex` using the plotly library.


RVI("SPY", 20, MovingAverageType.Simple)
variable
relativevigorindex
value
0.3
0.2
0.1
0
−0.1
−0.2
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Schaff Trend Cycle

Introduction

This indicator creates the Schaff Trend Cycle

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STC Indicator

To create an automatic indicators for `SchaffTrendCycle` , call the `STC` helper method from the `QCAlgorithm` class. The `STC` method creates a `SchaffTrendCycle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class SchaffTrendCycleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.stc = self.STC(self.symbol, 5, 10, 20, MovingAverageType.Exponential)

    def OnData(self, slice: Slice) -> None:
        if self.stc.IsReady:
            # The current value of self.stc is represented by self.stc.Current.Value
            self.Plot("SchaffTrendCycle", "stc", self.stc.Current.Value)
```

The following reference table describes the `STC` method:

INDICATORS

STC() 1/1

```
SchaffTrendCycle QuantConnect.Algorithm.QCAlgorithm.STC (
Symbol symbol,
Int32 cyclePeriod,
Int32 fastPeriod,
Int32 slowPeriod,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Schaff Trend Cycle indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol for the indicator to track.
`Int32`	cyclePeriod	The signal period.
`Int32`	fastPeriod	The fast moving average period.
`Int32`	slowPeriod	The slow moving average period.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SchaffTrendCycle` - The SchaffTrendCycle indicator for the requested symbol over the specified period.

Definition at line 1549 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `SchaffTrendCycle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class SchaffTrendCycleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.stc = SchaffTrendCycle(5, 10, 20, MovingAverageType.Exponential)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.stc.Update(bar.EndTime, bar.Close)

        if self.stc.IsReady:
            # The current value of self.stc is represented by self.stc.Current.Value
            self.Plot("SchaffTrendCycle", "stc", self.stc.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SchaffTrendCycleAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.stc = SchaffTrendCycle(5, 10, 20, MovingAverageType.Exponential)
        self.RegisterIndicator(self.symbol, self.stc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.stc.IsReady:
            # The current value of self.stc is represented by self.stc.Current.Value
            self.Plot("SchaffTrendCycle", "stc", self.stc.Current.Value)
```

The following reference table describes the `SchaffTrendCycle` constructor:

INDICATORS

SchaffTrendCycle() 1/2

```
SchaffTrendCycle QuantConnect.Indicators.SchaffTrendCycle (
*int fastPeriod,
*int slowPeriod,
*int cyclePeriod,
*MovingAverageType type
)
```

https://www.tradingpedia.com/forex-trading-indicators/schaff-trend-cycle.

Show Details

Parameters		
*int	fastPeriod	*(Optional) (Optional)* The fast moving average period. Default: 23.
*int	slowPeriod	*(Optional) (Optional)* The slow moving average period. Default: 50.
*int	cyclePeriod	*(Optional) (Optional)* The signal period. Default: 10.
*MovingAverageType	type	*(Optional) (Optional)* The type of moving averages to use. Default: MovingAverageType.Exponential.

Return

SchaffTrendCycle - The new SchaffTrendCycle indicator object.

Definition at line 58 of file Indicators/SchaffTrendCycle.cs.

INDICATORS

SchaffTrendCycle() 2/2

```
SchaffTrendCycle QuantConnect.Indicators.SchaffTrendCycle (
 string name,
 int fastPeriod,
 int slowPeriod,
 int cyclePeriod,
 MovingAverageType type
 )
```

Creates a new schaff trend cycle with the specified parameters.

Show Details

Parameters		
string	name	The name of this indicator.
int	fastPeriod	The fast moving average period.
int	slowPeriod	The slow moving average period.
int	cyclePeriod	The signal period.
MovingAverageType	type	The type of moving averages to use.

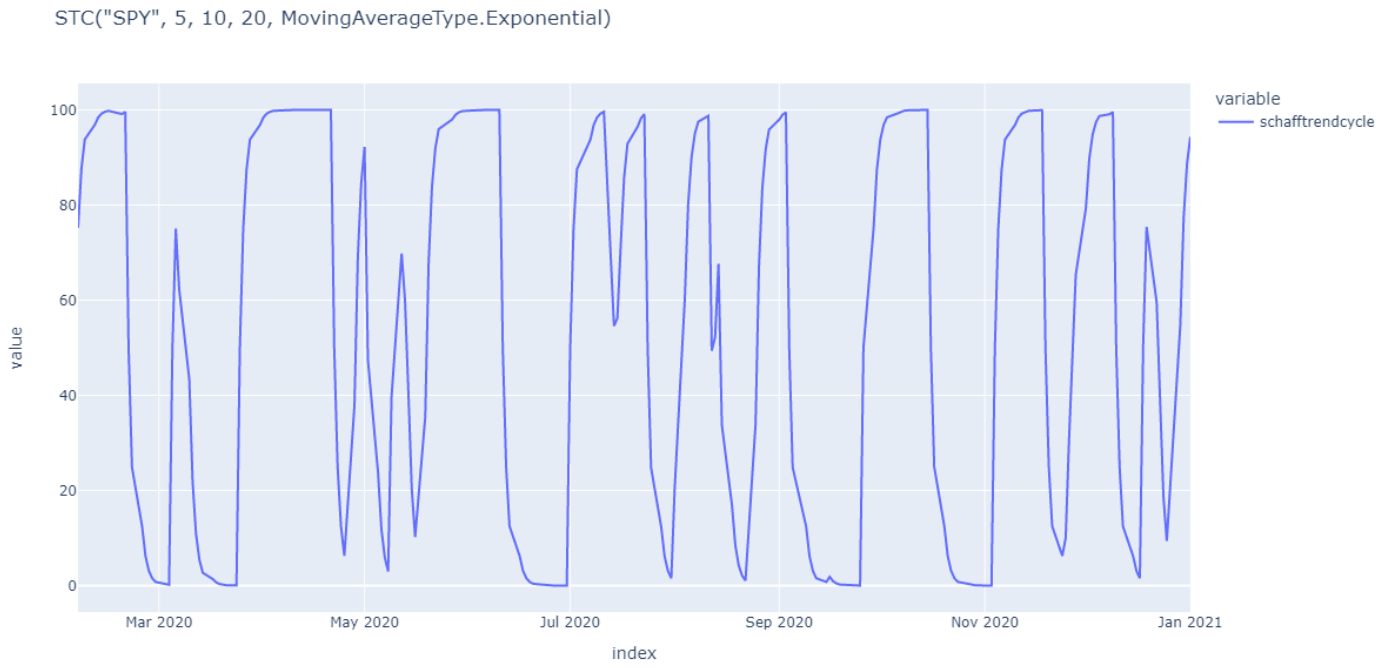
Return

`SchaffTrendCycle` - The new `SchaffTrendCycle` indicator object.

Definition at line 71 of file Indicators/SchaffTrendCycle.cs.

Visualization

The following image shows plot values of selected properties of `SchaffTrendCycle` using the plotly library.



Supported Indicators

Sharpe Ratio

Introduction

Calculation of the Sharpe Ratio (SR) developed by William F. Sharpe. Reference: https://www.investopedia.com/articles/07/sharpe_ratio.asp Formula: S(x) = (Rx - Rf) / stdDev(Rx) Where: S(x) - sharpe ratio of x Rx - average rate of return for x Rf - risk-free rate

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SR Indicator

To create an automatic indicators for `SharpeRatio` , call the `SR` helper method from the `QCAlgorithm` class. The `SR` method creates a `SharpeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class SharpeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sr = self.SR(self.symbol, 22, 0.03)

    def OnData(self, slice: Slice) -> None:
        if self.sr.IsReady:
            # The current value of self.sr is represented by self.sr.Current.Value
            self.Plot("SharpeRatio", "sr", self.sr.Current.Value)
```

The following reference table describes the `SR` method:

INDICATORS

SR() 1/1

```
SharpeRatio QuantConnect.Algorithm.QCAlgorithm.SR (
Symbol symbol,
Int32 sharpePeriod,
*Decimal riskFreeRate,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RollingSharpeRatio indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RSR we want.
`Int32`	sharpePeriod	Period of historical observation for sharpe ratio calculation.
`*Decimal`	riskFreeRate	*(Optional)* Risk-free rate for sharpe ratio calculation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SharpeRatio` - The RollingSharpeRatio indicator for the requested symbol over the specified period.

Definition at line 1489 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `SharpeRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class SharpeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sr = SharpeRatio(22, 0.03)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.sr.Update(bar.EndTime, bar.Close)

        if self.sr.IsReady:
            # The current value of self.sr is represented by self.sr.Current.Value
            self.Plot("SharpeRatio", "sr", self.sr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SharpeRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sr = SharpeRatio(22, 0.03)
        self.RegisterIndicator(self.symbol, self.sr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.sr.IsReady:
            # The current value of self.sr is represented by self.sr.Current.Value
            self.Plot("SharpeRatio", "sr", self.sr.Current.Value)
```

The following reference table describes the SharpeRatio constructor:

INDICATORS

SharpeRatio() 1/2

```
SharpeRatio QuantConnect.Indicators.SharpeRatio (
string name,
int period,
*decimal riskFreeRate
)
```

Creates a new Sharpe Ratio indicator using the specified periods.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	Period of historical observation for sharpe ratio calculation.
*decimal	riskFreeRate	*(Optional) (Optional)* Risk-free rate for sharpe ratio calculation. Default: 0.0m.

Return

SharpeRatio - The new SharpeRatio indicator object.

Definition at line 66 of file Indicators/SharpeRatio.cs.

INDICATORS

SharpeRatio() 2/2

```
SharpeRatio QuantConnect.Indicators.SharpeRatio (
int period,
*decimal riskFreeRate
)
```

Creates a new SharpeRatio indicator using the specified periods.

Show Details

Parameters		
int	period	Period of historical observation for sharpe ratio calculation.
*decimal	riskFreeRate	*(Optional) (Optional)* Risk-free rate for sharpe ratio calculation. Default: 0.0m.

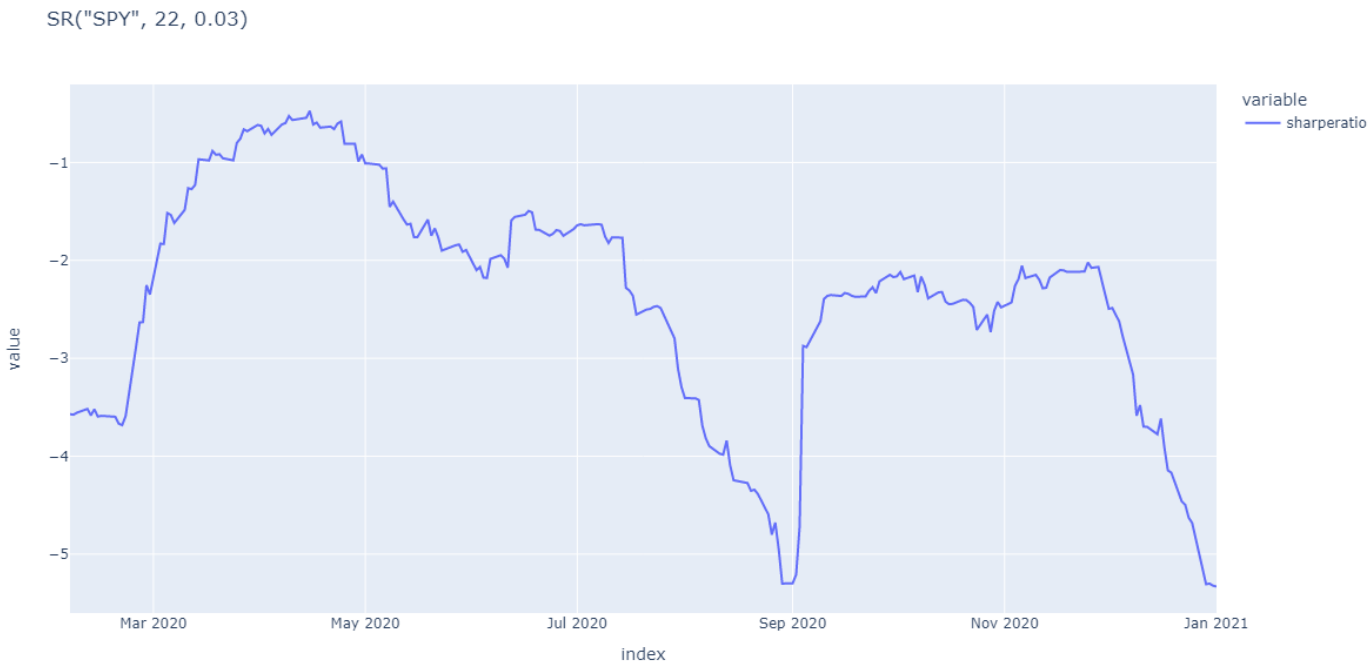
Return

SharpeRatio - The new SharpeRatio indicator object.

Definition at line 87 of file Indicators/SharpeRatio.cs.

Visualization

The following image shows plot values of selected properties of SharpeRatio using the plotly library.



Supported Indicators

Simple Moving Average

Introduction

This indicator represents the traditional simple moving average indicator (SMA)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SMA Indicator

To create an automatic indicators for `SimpleMovingAverage` , call the `SMA` helper method from the `QCAlgorithm` class. The `SMA` method creates a `SimpleMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class SimpleMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sma = self.SMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.sma.IsReady:
            # The current value of self.sma is represented by self.sma.Current.Value
            self.Plot("SimpleMovingAverage", "sma", self.sma.Current.Value)
            # Plot all attributes of self.sma
            self.Plot("SimpleMovingAverage", "rollingsum", self.sma.RollingSum.Current.Value)
```

The following reference table describes the `SMA` method:

INDICATORS

SMA() 1/1

```
SimpleMovingAverage QuantConnect.Algorithm.QCAlgorithm.SMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an SimpleMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose SMA we want.
`Int32`	period	The period of the SMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SimpleMovingAverage` - The SimpleMovingAverage for the given parameters.

Definition at line 1528 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `SimpleMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class SimpleMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sma = SimpleMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.sma.Update(bar.EndTime, bar.Close)

        if self.sma.IsReady:
            # The current value of self.sma is represented by self.sma.Current.Value
            self.Plot("SimpleMovingAverage", "sma", self.sma.Current.Value)
            # Plot all attributes of self.sma
            self.Plot("SimpleMovingAverage", "rollingsum", self.sma.RollingSum.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SimpleMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sma = SimpleMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.sma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.sma.IsReady:
            # The current value of self.sma is represented by self.sma.Current.Value
            self.Plot("SimpleMovingAverage", "sma", self.sma.Current.Value)
            # Plot all attributes of self.sma
            self.Plot("SimpleMovingAverage", "rollingsum", self.sma.RollingSum.Current.Value)
```

The following reference table describes the `SimpleMovingAverage` constructor:

INDICATORS

SimpleMovingAverage() 1/2

```
SimpleMovingAverage QuantConnect.Indicators.SimpleMovingAverage (
string name,
int period
)
```

Initializes a new instance of the SimpleMovingAverage class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the SMA.

Return

`SimpleMovingAverage` - The new `SimpleMovingAverage` indicator object.

Definition at line 47 of file Indicators/SimpleMovingAverage.cs.

INDICATORS

SimpleMovingAverage() 2/2

```
SimpleMovingAverage QuantConnect.Indicators.SimpleMovingAverage (
int period
)
```

Initializes a new instance of the SimpleMovingAverage class with the default name and period.

Show Details

Parameters		
int	period	The period of the SMA.

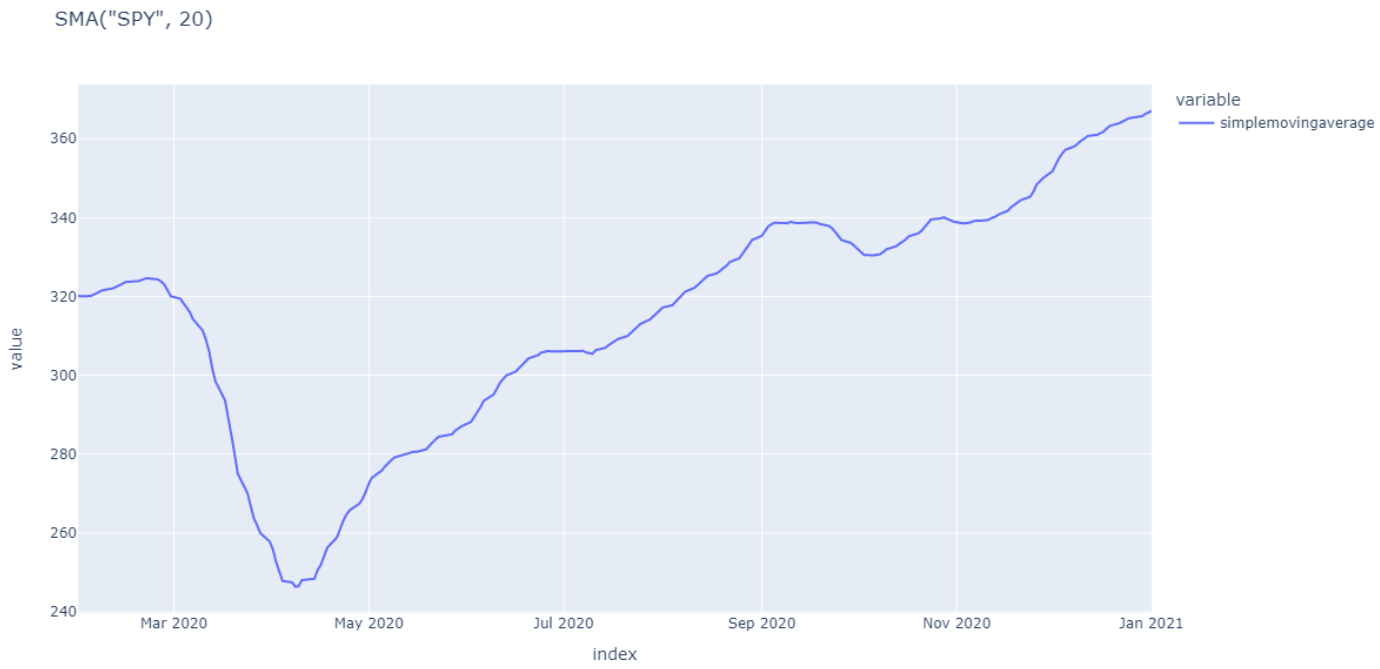
Return

`SimpleMovingAverage` - The new `SimpleMovingAverage` indicator object.

Definition at line 57 of file Indicators/SimpleMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `SimpleMovingAverage` using the plotly library.



Supported Indicators

Sortino Ratio

Introduction

Calculation of the Sortino Ratio, a modification of the . Reference: https://www.cmegroup.com/education/files/rr-sortino-a-sharper-ratio.pdf Formula: S(x) = (R - T) / TDD Where: S(x) - Sortino ratio of x R - the average period return T - the target or required rate of return for the investment strategy under consideration. In Sortino's early work, T was originally known as the minimum acceptable return, or MAR. In his more recent work, MAR is now referred to as the Desired Target Return. TDD - the target downside deviation.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SORTINO Indicator

To create an automatic indicators for `SortinoRatio` , call the `SORTINO` helper method from the `QCAlgorithm` class. The `SORTINO` method creates a `SortinoRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class SortinoRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sortino = self.SORTINO(self.symbol, 22)

    def OnData(self, slice: Slice) -> None:
        if self.sortino.IsReady:
            # The current value of self.sortino is represented by self.sortino.Current.Value
            self.Plot("SortinoRatio", "sortino", self.sortino.Current.Value)
```

The following reference table describes the `SORTINO` method:

INDICATORS

SORTINO() 1/1

```
SortinoRatio QuantConnect.Algorithm.QCAlgorithm.SORTINO (
Symbol symbol,
Int32 sortinoPeriod,
*Double minimumAcceptableReturn,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Sortino indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Sortino we want.
`Int32`	sortinoPeriod	Period of historical observation for Sortino ratio calculation.
`*Double`	minimumAcceptableReturn	*(Optional)* Minimum acceptable return (eg risk-free rate) for the Sortino ratio calculation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SortinoRatio` - The SortinoRatio indicator for the requested symbol over the specified period.

Definition at line 1508 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `SortinoRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class SortinoRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sortino = SortinoRatio(22)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.sortino.Update(bar.EndTime, bar.Close)

        if self.sortino.IsReady:
            # The current value of self.sortino is represented by self.sortino.Current.Value
            self.Plot("SortinoRatio", "sortino", self.sortino.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SortinoRatioAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sortino = SortinoRatio(22)
        self.RegisterIndicator(self.symbol, self.sortino, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.sortino.IsReady:
            # The current value of self.sortino is represented by self.sortino.Current.Value
            self.Plot("SortinoRatio", "sortino", self.sortino.Current.Value)
```

The following reference table describes the `SortinoRatio` constructor:

INDICATORS

SortinoRatio() 1/2

```
SortinoRatio QuantConnect.Indicators.SortinoRatio (
string name,
int period,
*double minimumAcceptableReturn
)
```

Creates a new Sortino Ratio indicator using the specified periods.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	Period of historical observation for Sortino ratio calculation.
*double	minimumAcceptableReturn	*(Optional)* *(Optional)* Minimum acceptable return for Sortino ratio calculation. Default: 0.

Return

`SortinoRatio` - The new `SortinoRatio` indicator object.

Definition at line 39 of file Indicators/SortinoRatio.cs.

INDICATORS

SortinoRatio() 2/2

```
SortinoRatio QuantConnect.Indicators.SortinoRatio (
int period,
*double minimumAcceptableReturn
)
```

Creates a new SortinoRatio indicator using the specified periods.

Show Details

Parameters		
int	period	Period of historical observation for Sortino ratio calculation.
*double	minimumAcceptableReturn	*(Optional)* *(Optional)* Minimum acceptable return for Sortino ratio calculation. Default: 0.

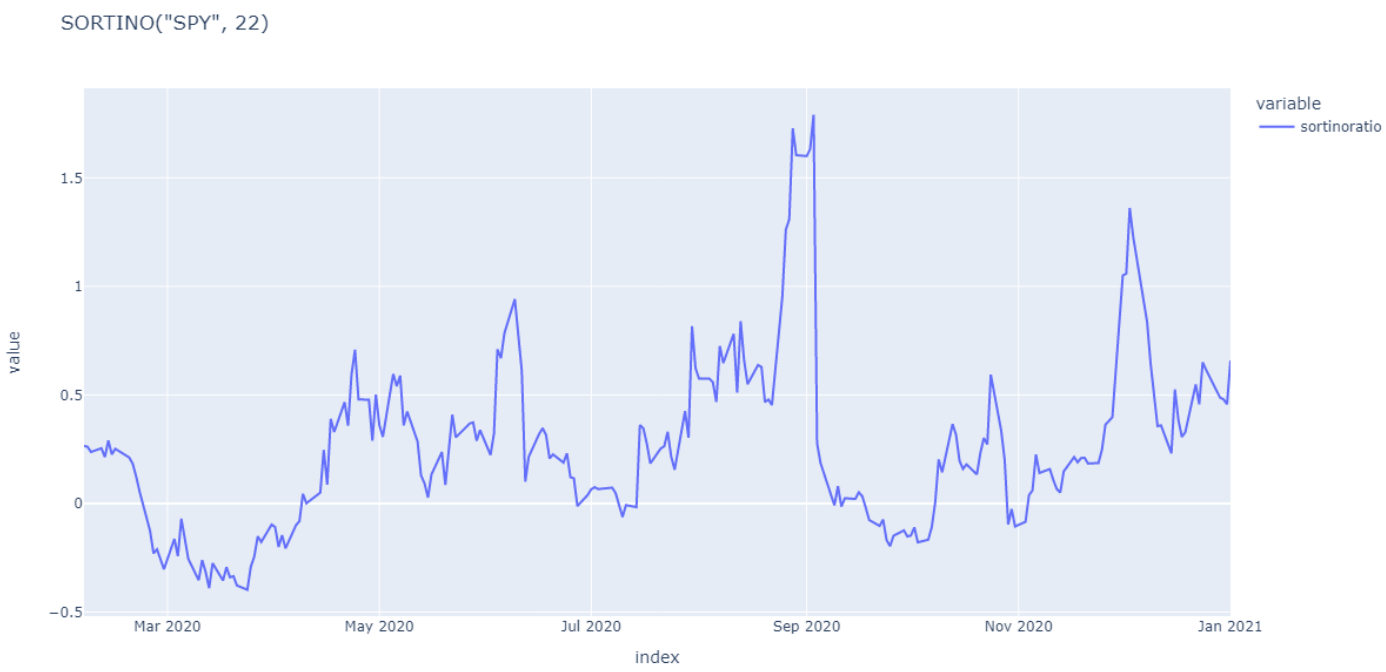
Return

`SortinoRatio` - The new `SortinoRatio` indicator object.

Definition at line 51 of file Indicators/SortinoRatio.cs.

Visualization

The following image shows plot values of selected properties of `SortinoRatio` using the plotly library.



Supported Indicators

Standard Deviation

Introduction

This indicator computes the n-period population standard deviation.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STD Indicator

To create an automatic indicators for `StandardDeviation` , call the `STD` helper method from the `QCAlgorithm` class. The `STD` method creates a `StandardDeviation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class StandardDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.std = self.STD(self.symbol, 22)

    def OnData(self, slice: Slice) -> None:
        if self.std.IsReady:
            # The current value of self.std is represented by self.std.Current.Value
            self.Plot("StandardDeviation", "std", self.std.Current.Value)
```

The following reference table describes the `STD` method:

INDICATORS

STD() 1/1

```
StandardDeviation QuantConnect.Algorithm.QCAlgorithm.STD (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new StandardDeviation indicator. This will return the population standard deviation of samples over the specified period.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose STD we want.
`Int32`	period	The period over which to compute the STD.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`StandardDeviation` - The StandardDeviation indicator for the requested symbol over the specified period.

Definition at line 1567 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `StandardDeviation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class StandardDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.std = StandardDeviation(22)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.std.Update(bar.EndTime, bar.Close)

        if self.std.IsReady:
            # The current value of self.std is represented by self.std.Current.Value
            self.Plot("StandardDeviation", "std", self.std.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class StandardDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.std = StandardDeviation(22)
        self.RegisterIndicator(self.symbol, self.std, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.std.IsReady:
            # The current value of self.std is represented by self.std.Current.Value
            self.Plot("StandardDeviation", "std", self.std.Current.Value)
```

The following reference table describes the `StandardDeviation` constructor:

INDICATORS

StandardDeviation() 1/2

```
StandardDeviation QuantConnect.Indicators.StandardDeviation (
int period
)
```

On a data set of size N will use an N normalizer and would thus be biased if applied to a subset.

Show Details

Parameters		
int	period	The sample size of the standard deviation.

Return

`StandardDeviation` - The new `StandardDeviation` indicator object.

Definition at line 32 of file Indicators/StandardDeviation.cs.

INDICATORS

StandardDeviation() 2/2

```
StandardDeviation QuantConnect.Indicators.StandardDeviation (
string name,
int period
)
```

On a data set of size N will use an N normalizer and would thus be biased if applied to a subset.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The sample size of the standard deviation.

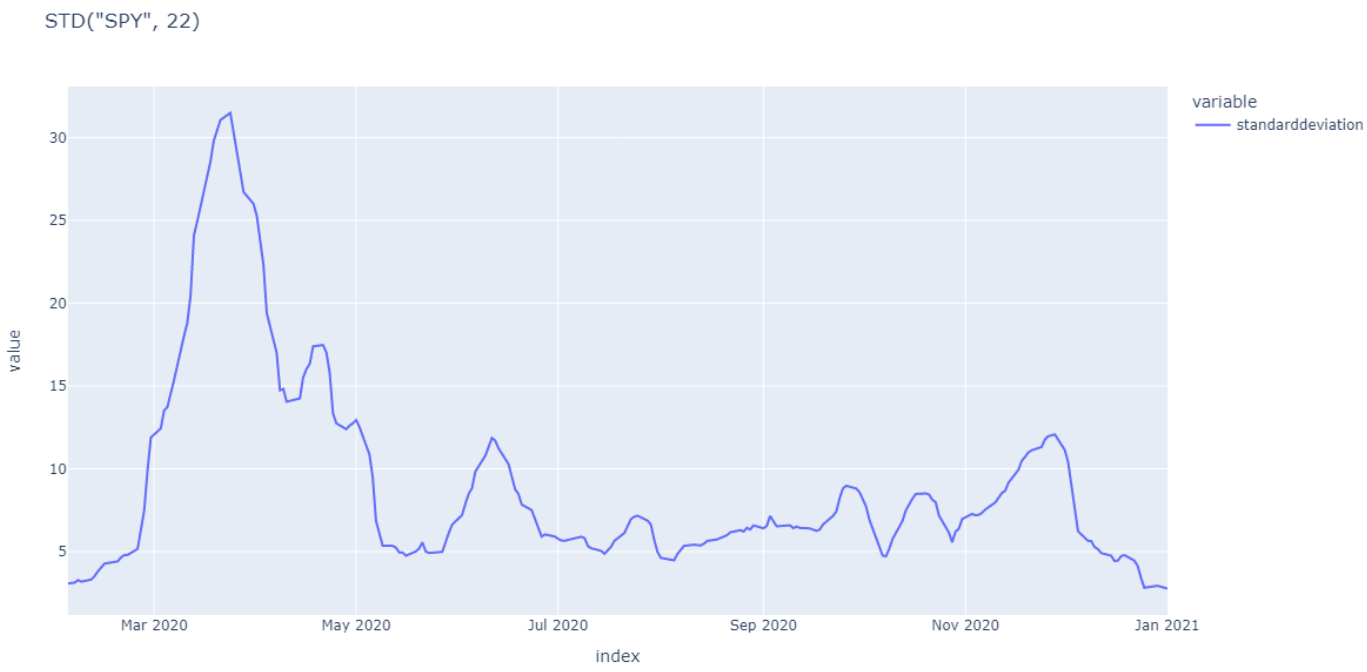
Return

`StandardDeviation` - The new `StandardDeviation` indicator object.

Definition at line 45 of file Indicators/StandardDeviation.cs.

Visualization

The following image shows plot values of selected properties of `StandardDeviation` using the plotly library.



Supported Indicators

Stochastic

Introduction

This indicator computes the Slow Stochastics %K and %D. The Fast Stochastics %K is is computed by (Current Close Price - Lowest Price of given Period) / (Highest Price of given Period - Lowest Price of given Period) multiplied by 100. Once the Fast Stochastics %K is calculated the Slow Stochastic %K is calculated by the average/smoothed price of of the Fast %K with the given period. The Slow Stochastics %D is then derived from the Slow Stochastics %K with the given period.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STO Indicator

To create an automatic indicators for `Stochastic` , call the `STO` helper method from the `QCAlgorithm` class. The `STO` method creates a `Stochastic` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class StochasticAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sto = self.STO(self.symbol, 20, 10, 20)

    def OnData(self, slice: Slice) -> None:
        if self.sto.IsReady:
            # The current value of self.sto is represented by self.sto.Current.Value
            self.Plot("Stochastic", "sto", self.sto.Current.Value)
            # Plot all attributes of self.sto
            self.Plot("Stochastic", "faststoch", self.sto.FastStoch.Current.Value)
            self.Plot("Stochastic", "stochk", self.sto.StochK.Current.Value)
            self.Plot("Stochastic", "stochd", self.sto.StochD.Current.Value)
```

The following reference table describes the `STO` method:

INDICATORS

STO() 1/2

```
Stochastic QuantConnect.Algorithm.QCAlgorithm.STO (
Symbol symbol,
Int32 period,
Int32 kPeriod,
Int32 dPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Stochastic indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose stochastic we seek.
`Int32`	period	The period of the stochastic. Normally 14.
`Int32`	kPeriod	The sum period of the stochastic. Normally 14.
`Int32`	dPeriod	The sum period of the stochastic. Normally 3.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`Stochastic` - Stochastic indicator for the requested symbol.

Definition at line 1607 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

STO() 2/2

```
Stochastic QuantConnect.Algorithm.QCAlgorithm.STO (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Overload short hand to create a new Stochastic indicator; defaulting to the 3 period for dStoch.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose stochastic we seek.
`Int32`	period	The period of the stochastic. Normally 14.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`Stochastic` - Stochastic indicator for the requested symbol.

Definition at line 1626 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Stochastic` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class StochasticAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sto = Stochastic(20, 10, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.sto.Update(bar)

        if self.sto.IsReady:
            # The current value of self.sto is represented by self.sto.Current.Value
            self.Plot("Stochastic", "sto", self.sto.Current.Value)
            # Plot all attributes of self.sto
            self.Plot("Stochastic", "faststoch", self.sto.FastStoch.Current.Value)
            self.Plot("Stochastic", "stochk", self.sto.StochK.Current.Value)
            self.Plot("Stochastic", "stochd", self.sto.StochD.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class StochasticAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sto = Stochastic(20, 10, 20)
        self.RegisterIndicator(self.symbol, self.sto, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.sto.IsReady:
            # The current value of self.sto is represented by self.sto.Current.Value
            self.Plot("Stochastic", "sto", self.sto.Current.Value)
            # Plot all attributes of self.sto
            self.Plot("Stochastic", "faststoch", self.sto.FastStoch.Current.Value)
            self.Plot("Stochastic", "stochk", self.sto.StochK.Current.Value)
            self.Plot("Stochastic", "stochd", self.sto.StochD.Current.Value)
```

The following reference table describes the `Stochastic` constructor:

INDICATORS

Stochastic() 1/2

```
Stochastic QuantConnect.Indicators.Stochastic (
string name,
int period,
int kPeriod,
int dPeriod
)
```

Creates a new Stochastics Indicator from the specified periods.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period given to calculate the Fast %K.
int	kPeriod	The K period given to calculated the Slow %K.
int	dPeriod	The D period given to calculated the Slow %D.

Return

`Stochastic` - The new `Stochastic` indicator object.

Definition at line 56 of file Indicators/Stochastics.cs.

INDICATORS

Stochastic() 2/2

```
Stochastic QuantConnect.Indicators.Stochastic (
int period,
int kPeriod,
int dPeriod
)
```

Creates a new `Stochasti` indicator from the specified inputs.

Show Details

Parameters		
`int`	period	The period given to calculate the Fast %K.
`int`	kPeriod	The K period given to calculated the Slow %K.
`int`	dPeriod	The D period given to calculated the Slow %D.

Return

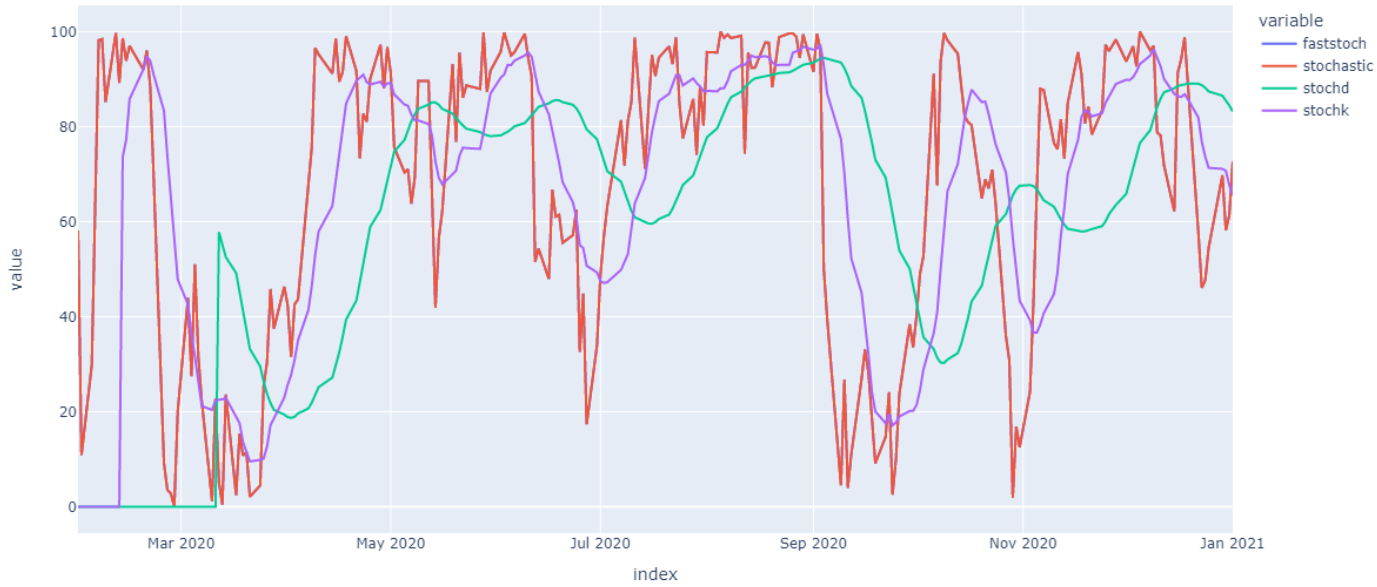
`Stochastic` - The new `Stochastic` indicator object.

Definition at line 92 of file Indicators/Stochastics.cs.

Visualization

The following image shows plot values of selected properties of `Stochastic` using the plotly library.

STO("SPY", 20, 10, 20)



Supported Indicators

Sum

Introduction

This indicator represents an indicator capable of tracking the sum for the given period

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SUM Indicator

To create an automatic indicators for `Sum` , call the `SUM` helper method from the `QCAlgorithm` class. The `SUM` method creates a `Sum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class SumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sum = self.SUM(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.sum.IsReady:
            # The current value of self.sum is represented by self.sum.Current.Value
            self.Plot("Sum", "sum", self.sum.Current.Value)
```

The following reference table describes the `SUM` method:

INDICATORS

SUM() 1/1

```
Sum QuantConnect.Algorithm.QCAlgorithm.SUM (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Sum indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Sum we want.
`Int32`	period	The period over which to compute the Sum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Sum` - The Sum indicator for the requested symbol over the specified period.

Definition at line 1640 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Sum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class SumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sum = Sum(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.sum.Update(bar.EndTime, bar.Close)

        if self.sum.IsReady:
            # The current value of self.sum is represented by self.sum.Current.Value
            self.Plot("Sum", "sum", self.sum.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class SumAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.sum = Sum(20)
        self.RegisterIndicator(self.symbol, self.sum, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.sum.IsReady:
            # The current value of self.sum is represented by self.sum.Current.Value
            self.Plot("Sum", "sum", self.sum.Current.Value)
```

The following reference table describes the `Sum` constructor:

INDICATORS

Sum() 1/2

```
Sum QuantConnect.Indicators.Sum (
string name,
int period
)
```

Initializes a new instance of the Sum class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the SMA.

Return

`Sum` - The new `Sum` indicator object.

Definition at line 47 of file Indicators/Sum.cs.

INDICATORS

Sum() 2/2

```
Sum QuantConnect.Indicators.Sum (
int period
)
```

Initializes a new instance of the Sum class with the default name and period.

Show Details

Parameters		
`int`	period	The period of the SMA.

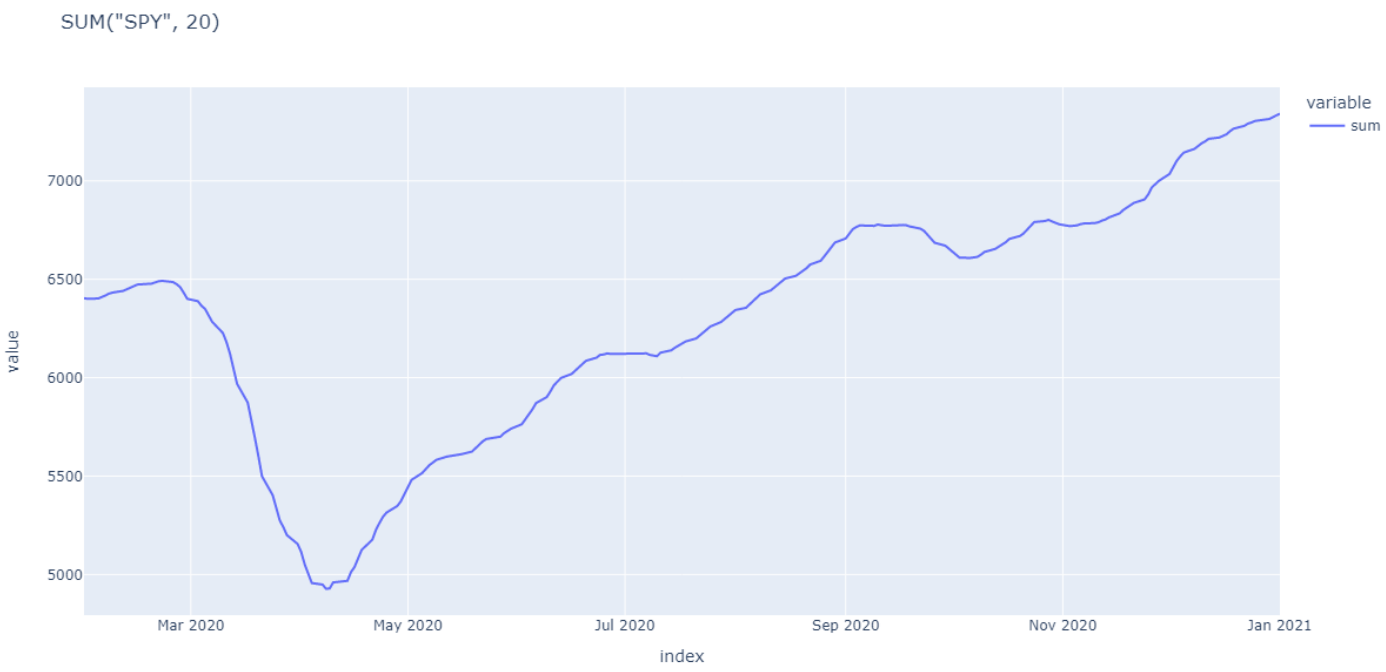
Return

`Sum` - The new `Sum` indicator object.

Definition at line 56 of file Indicators/Sum.cs.

Visualization

The following image shows plot values of selected properties of `Sum` using the plotly library.



Supported Indicators

Super Trend

Introduction

Super trend indicator. Formula can be found here via the excel file: https://tradingtuitions.com/supertrend-indicator-excel-sheet-with-realtime-buy-sell-signals/

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STR Indicator

To create an automatic indicators for `SuperTrend` , call the `STR` helper method from the `QCAlgorithm` class. The `STR` method creates a `SuperTrend` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class SuperTrendAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.str = self.STR(self.symbol, 20, 2, MovingAverageType.Wilders)

    def OnData(self, slice: Slice) -> None:
        if self.str.IsReady:
            # The current value of self.str is represented by self.str.Current.Value
            self.Plot("SuperTrend", "str", self.str.Current.Value)
            # Plot all attributes of self.str
            self.Plot("SuperTrend", "basicupperband", self.str.BasicUpperBand.Current.Value)
            self.Plot("SuperTrend", "basiclowerband", self.str.BasicLowerBand.Current.Value)
            self.Plot("SuperTrend", "currenttrailingupperband",
self.str.CurrentTrailingUpperBand.Current.Value)
            self.Plot("SuperTrend", "currenttrailinglowerband",
self.str.CurrentTrailingLowerBand.Current.Value)
```

The following reference table describes the `STR` method:

INDICATORS

STR() 1/1

```
SuperTrend QuantConnect.Algorithm.QCAlgorithm.STR (
Symbol symbol,
Int32 period,
Decimal multiplier,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new SuperTrend indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose SuperTrend indicator we want.
`Int32`	period	The smoothing period for average True range.
`Decimal`	multiplier	Multiplier to calculate basic upper and lower bands width.
`*MovingAverageType`	movingAverageType	*(Optional)* Smoother type for average True range, defaults to Wilders.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`SuperTrend` - The new `SuperTrend` object.

Definition at line 1469 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `SuperTrend` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

```py
class SuperTrendAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.str = SuperTrend(20, 2, MovingAverageType.Wilders)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.str.Update(bar)

        if self.str.IsReady:
            # The current value of self.str is represented by self.str.Current.Value
            self.Plot("SuperTrend", "str", self.str.Current.Value)
            # Plot all attributes of self.str
            self.Plot("SuperTrend", "basicupperband", self.str.BasicUpperBand.Current.Value)
            self.Plot("SuperTrend", "basiclowerband", self.str.BasicLowerBand.Current.Value)
            self.Plot("SuperTrend", "currenttrailingupperband",
self.str.CurrentTrailingUpperBand.Current.Value)
            self.Plot("SuperTrend", "currenttrailinglowerband",
self.str.CurrentTrailingLowerBand.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SuperTrendAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.str = SuperTrend(20, 2, MovingAverageType.Wilders)
        self.RegisterIndicator(self.symbol, self.str, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.str.IsReady:
            # The current value of self.str is represented by self.str.Current.Value
            self.Plot("SuperTrend", "str", self.str.Current.Value)
            # Plot all attributes of self.str
            self.Plot("SuperTrend", "basicupperband", self.str.BasicUpperBand.Current.Value)
            self.Plot("SuperTrend", "basiclowerband", self.str.BasicLowerBand.Current.Value)
            self.Plot("SuperTrend", "currenttrailingupperband",
self.str.CurrentTrailingUpperBand.Current.Value)
            self.Plot("SuperTrend", "currenttrailinglowerband",
self.str.CurrentTrailingLowerBand.Current.Value)
```

The following reference table describes the `SuperTrend` constructor:

INDICATORS

SuperTrend() 1/2

```
SuperTrend QuantConnect.Indicators.SuperTrend (
string name,
int period,
decimal multiplier,
*MovingAverageType movingAverageType
)
```

Creates a new SuperTrend indicator using the specified name, period, multiplier and moving average type.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The smoothing period used by average true range.
decimal	multiplier	The coefficient used in calculations of basic upper and lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of smoothing used to smooth the true range values. Default: MovingAverageType.Wilders.

Return

SuperTrend - The new SuperTrend indicator object.

Definition at line 78 of file Indicators/SuperTrend.cs.

INDICATORS

SuperTrend() 2/2

```
SuperTrend QuantConnect.Indicators.SuperTrend (
int period,
decimal multiplier,
*MovingAverageType movingAverageType
)
```

Creates a new SuperTrend indicator using the specified period, multiplier and moving average type.

Show Details

Parameters		
int	period	The smoothing period used in average true range.
decimal	multiplier	The coefficient used in calculations of basic upper and lower bands.
*MovingAverageType	movingAverageType	*(Optional) (Optional)* The type of smoothing used to smooth the true range values. Default: MovingAverageType.Wilders.

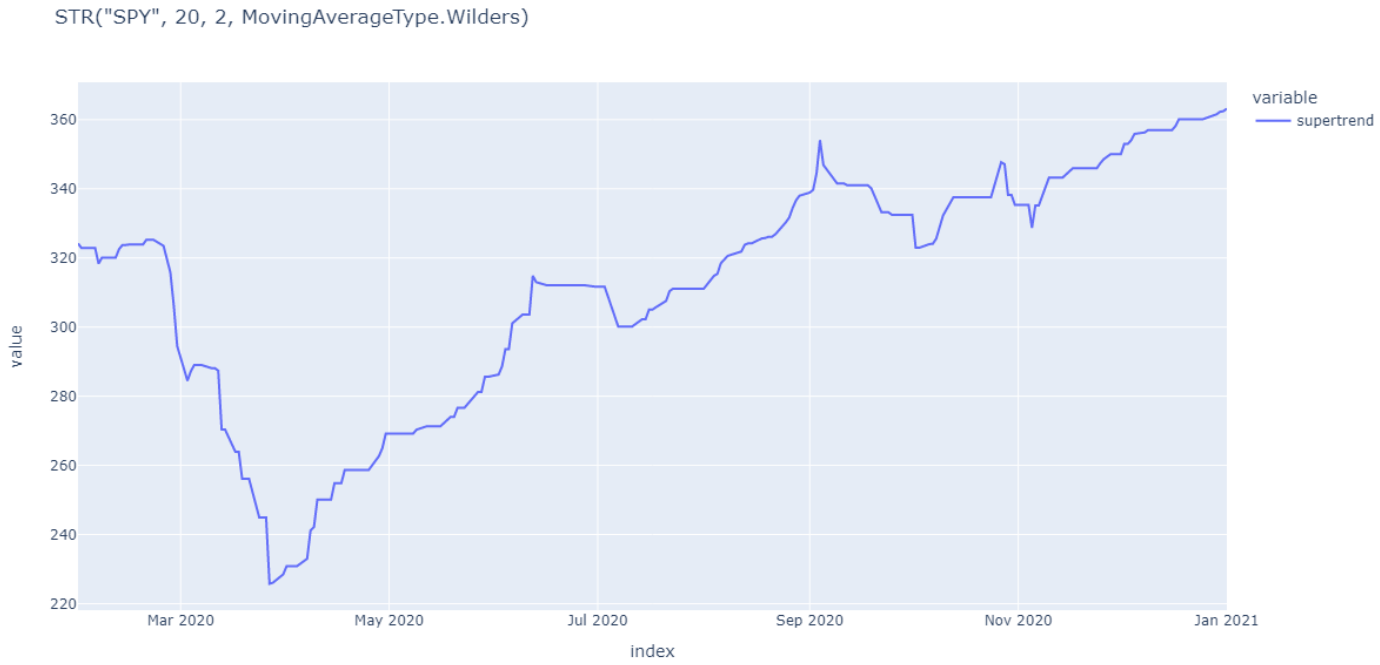
Return

SuperTrend - The new SuperTrend indicator object.

Definition at line 93 of file Indicators/SuperTrend.cs.

Visualization

The following image shows plot values of selected properties of `SuperTrend` using the plotly library.



Supported Indicators

Swiss Army Knife

Introduction

Swiss Army Knife indicator by John Ehlers

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SWISS Indicator

To create an automatic indicators for `SwissArmyKnife` , call the `SWISS` helper method from the `QCAlgorithm` class. The `SWISS` method creates a `SwissArmyKnife` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class SwissArmyKnifeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.swiss = self.SWISS(self.symbol, 20, 0.2, SwissArmyKnifeTool.Gauss)

    def OnData(self, slice: Slice) -> None:
        if self.swiss.IsReady:
            # The current value of self.swiss is represented by self.swiss.Current.Value
            self.Plot("SwissArmyKnife", "swiss", self.swiss.Current.Value)
```

The following reference table describes the `SWISS` method:

INDICATORS

SWISS() 1/1

```
SwissArmyKnife QuantConnect.Algorithm.QCAlgorithm.SWISS (
Symbol symbol,
Int32 period,
Double delta,
SwissArmyKnifeTool tool,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates Swiss Army Knife transformation for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol to use for calculations.
`Int32`	period	The period of the calculation.
`Double`	delta	The delta scale of the BandStop or BandPass.
`SwissArmyKnifeTool`	tool	The tool os the Swiss Army Knife. *Options: ['Gauss', 'Butter', 'HighPass', 'TwoPoleHighPass', 'BandPass']*
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* elects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SwissArmyKnife` - The calculation using the given tool.

Definition at line 1661 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `SwissArmyKnife` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class SwissArmyKnifeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.swiss = SwissArmyKnife(20, 0.2, SwissArmyKnifeTool.Gauss)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.swiss.Update(bar.EndTime, bar.Close)

        if self.swiss.IsReady:
            # The current value of self.swiss is represented by self.swiss.Current.Value
            self.Plot("SwissArmyKnife", "swiss", self.swiss.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class SwissArmyKnifeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.swiss = SwissArmyKnife(20, 0.2, SwissArmyKnifeTool.Gauss)
        self.RegisterIndicator(self.symbol, self.swiss, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.swiss.IsReady:
            # The current value of self.swiss is represented by self.swiss.Current.Value
            self.Plot("SwissArmyKnife", "swiss", self.swiss.Current.Value)
```

The following reference table describes the `SwissArmyKnife` constructor:

INDICATORS

SwissArmyKnife() 1/2

```
SwissArmyKnife QuantConnect.Indicators.SwissArmyKnife (
int period,
double delta,
SwissArmyKnifeTool tool
)
```

Swiss Army Knife indicator by John Ehlers.

Show Details

Parameters		
int	period	/
double	delta	/
SwissArmyKnifeTool	tool	/

Return

`SwissArmyKnife` - The new `SwissArmyKnife` indicator object.

Definition at line 69 of file Indicators/SwissArmyKnife.cs.

INDICATORS

SwissArmyKnife() 2/2

```
SwissArmyKnife QuantConnect.Indicators.SwissArmyKnife (
string name,
int period,
double delta,
SwissArmyKnifeTool tool
)
```

Swiss Army Knife indicator by John Ehlers.

Show Details

Parameters		
`string`	name	/
`int`	period	/
`double`	delta	/
`SwissArmyKnifeTool`	tool	/

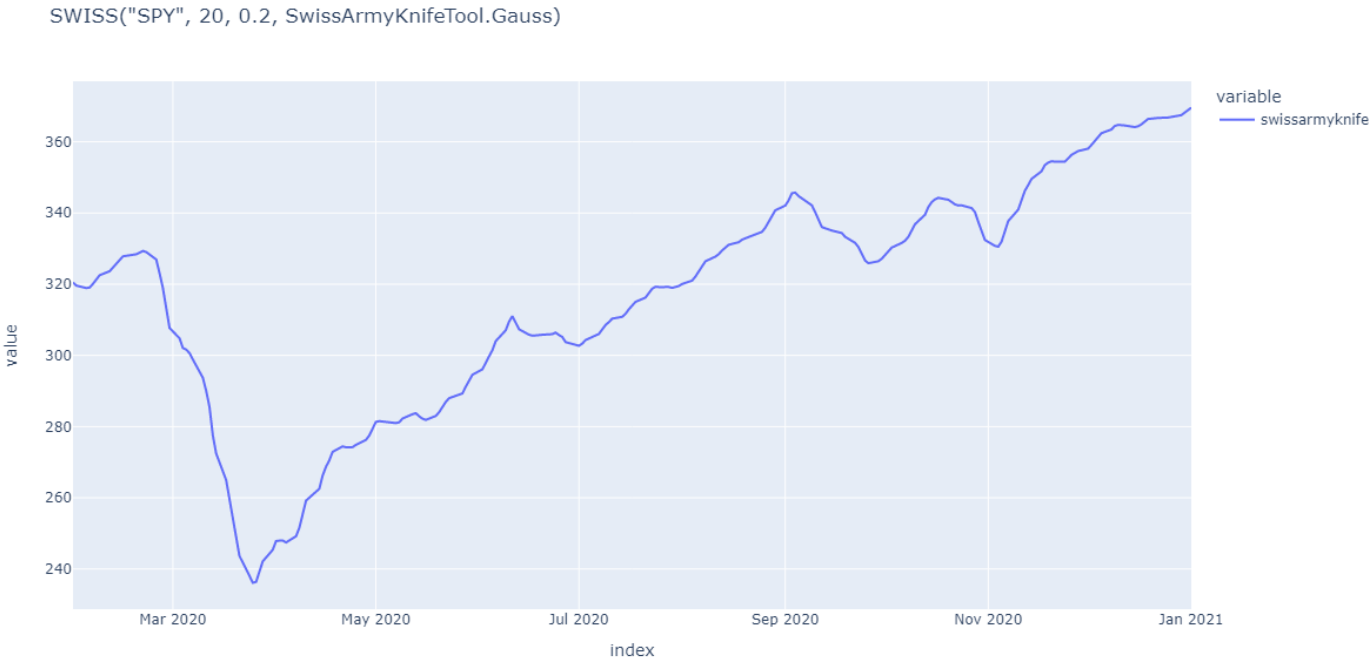
Return

`SwissArmyKnife` - The new `SwissArmyKnife` indicator object.

Definition at line 81 of file Indicators/SwissArmyKnife.cs.

Visualization

The following image shows plot values of selected properties of `SwissArmyKnife` using the plotly library.


SWISS("SPY", 20, 0.2, SwissArmyKnifeTool.Gauss)
variable
swissarmyknife
360
340
320
300
280
260
240
value
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

T3 Moving Average

Introduction

This indicator computes the T3 Moving Average (T3). The T3 Moving Average is calculated with the following formula: EMA1(x, Period) = EMA(x, Period) EMA2(x, Period) = EMA(EMA1(x, Period),Period) GD(x, Period, volumeFactor) = (EMA1(x, Period)*(1+volumeFactor)) - (EMA2(x, Period)* volumeFactor) T3 = GD(GD(GD(t, Period, volumeFactor), Period, volumeFactor), Period, volumeFactor);

To view the implementation of this indicator, see the LEAN GitHub repository .

Using T3 Indicator

To create an automatic indicators for `T3MovingAverage` , call the `T3` helper method from the `QCAlgorithm` class. The `T3` method creates a `T3MovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class T3MovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.t3 = self.T3(self.symbol, 30, 0.7)

    def OnData(self, slice: Slice) -> None:
        if self.t3.IsReady:
            # The current value of self.t3 is represented by self.t3.Current.Value
            self.Plot("T3MovingAverage", "t3", self.t3.Current.Value)
```

The following reference table describes the `T3` method:

INDICATORS

T3() 1/1

```
T3MovingAverage QuantConnect.Algorithm.QCAlgorithm.T3 (
Symbol symbol,
Int32 period,
*Decimal volumeFactor,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new T3MovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose T3 we want.
`Int32`	period	The period over which to compute the T3.
`*Decimal`	volumeFactor	*(Optional)* The volume factor to be used for the T3 (value must be in the [0,1] range, defaults to 0.7).
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`T3MovingAverage` - The T3MovingAverage indicator for the requested symbol over the specified period.

Definition at line 1680 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `T3MovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class T3MovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.t3 = T3MovingAverage(30, 0.7)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.t3.Update(bar.EndTime, bar.Close)

        if self.t3.IsReady:
            # The current value of self.t3 is represented by self.t3.Current.Value
            self.Plot("T3MovingAverage", "t3", self.t3.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class T3MovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.t3 = T3MovingAverage(30, 0.7)
        self.RegisterIndicator(self.symbol, self.t3, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.t3.IsReady:
            # The current value of self.t3 is represented by self.t3.Current.Value
            self.Plot("T3MovingAverage", "t3", self.t3.Current.Value)
```

The following reference table describes the `T3MovingAverage` constructor:

INDICATORS

T3MovingAverage() 1/2

```
T3MovingAverage QuantConnect.Indicators.T3MovingAverage (
string name,
int period,
*decimal volumeFactor
)
```

Initializes a new instance of the `T3MovingAverag` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the T3MovingAverage.
`*decimal`	volumeFactor	*(Optional) (Optional)* The volume factor of the T3MovingAverage (value must be in the [0,1] range, defaults to 0.7). Default: 0.7m.

Return

`T3MovingAverage` - The new `T3MovingAverage` indicator object.

Definition at line 39 of file Indicators/T3MovingAverage.cs.

INDICATORS

T3MovingAverage() 2/2

```
T3MovingAverage QuantConnect.Indicators.T3MovingAverage (
int period,
*decimal volumeFactor
)
```

Initializes a new instance of the `T3MovingAverag` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the T3MovingAverage.
`*decimal`	volumeFactor	*(Optional) (Optional)* The volume factor of the T3MovingAverage (value must be in the [0,1] range, defaults to 0.7). Default: 0.7m.

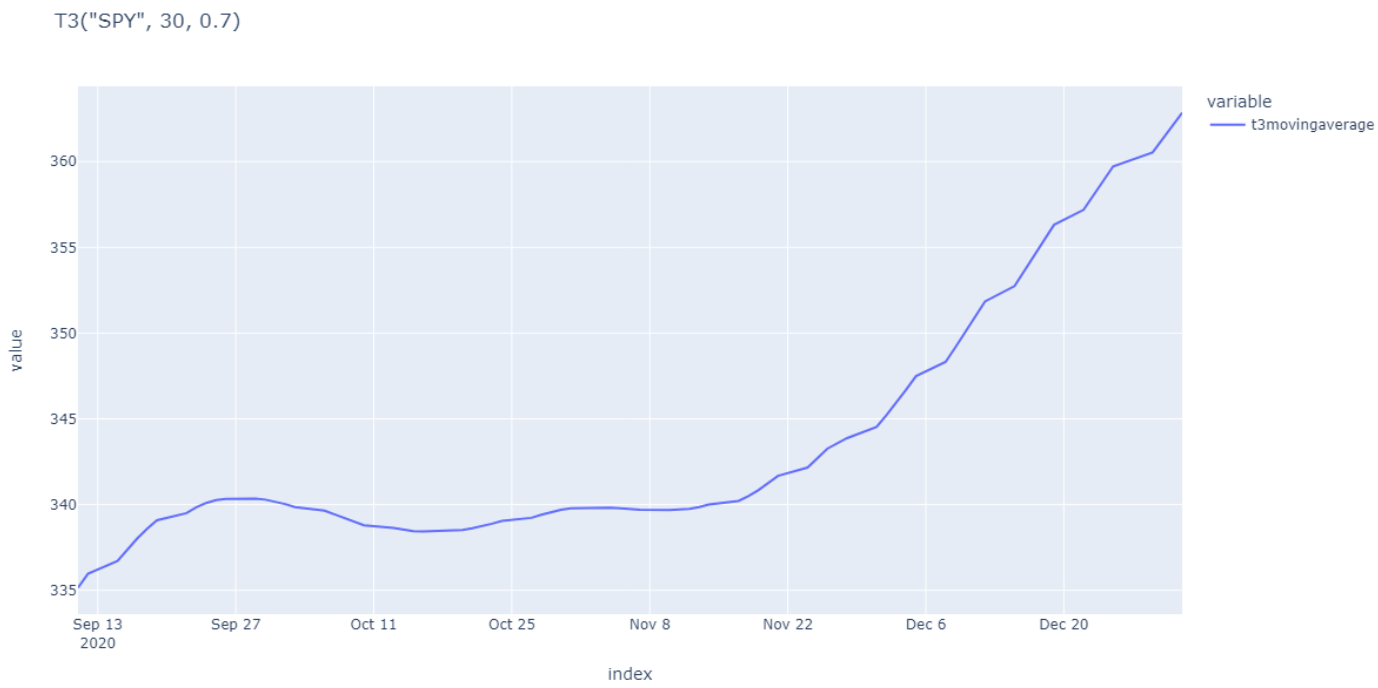
Return

`T3MovingAverage` - The new `T3MovingAverage` indicator object.

Definition at line 53 of file Indicators/T3MovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `T3MovingAverage` using the plotly library.



Supported Indicators

Target Downside Deviation

Introduction

This indicator computes the n-period target downside deviation. The target downside deviation is defined as the root-mean-square, or RMS, of the deviations of the realized return's underperformance from the target return where all returns above the target return are treated as underperformance of 0. Reference: https://www.cmegroup.com/education/files/rr-sortino-a-sharper-ratio.pdf

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TDD Indicator

To create an automatic indicators for `TargetDownsideDeviation` , call the `TDD` helper method from the `QCAlgorithm` class. The `TDD` method creates a `TargetDownsideDeviation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TargetDownsideDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tdd = self.TDD(self.symbol, 50)

    def OnData(self, slice: Slice) -> None:
        if self.tdd.IsReady:
            # The current value of self.tdd is represented by self.tdd.Current.Value
            self.Plot("TargetDownsideDeviation", "tdd", self.tdd.Current.Value)
```

The following reference table describes the `TDD` method:

INDICATORS

TDD() 1/1

```
TargetDownsideDeviation QuantConnect.Algorithm.QCAlgorithm.TDD (
Symbol symbol,
Int32 period,
*Double minimumAcceptableReturn,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new TargetDownsideDeviation indicator. The target downside deviation is defined as the root-mean-square, or RMS, of the deviations of the realized return's underperformance from the target return where all

returns above the target return are treated as underperformance of 0.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TDD we want.
`Int32`	period	The period over which to compute the TDD.
`*Double`	minimumAcceptableReturn	*(Optional)* Minimum acceptable return (MAR) for the target downside deviation calculation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TargetDownsideDeviation` - The TargetDownsideDeviation indicator for the requested symbol over the specified period.

Definition at line 1587 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TargetDownsideDeviation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class TargetDownsideDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tdd = TargetDownsideDeviation(TargetDownsideDeviation(50), RateOfChange(1))

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tdd.Update(bar.EndTime, bar.Close)

        if self.tdd.IsReady:
            # The current value of self.tdd is represented by self.tdd.Current.Value
            self.Plot("TargetDownsideDeviation", "tdd", self.tdd.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class TargetDownsideDeviationAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tdd = TargetDownsideDeviation(TargetDownsideDeviation(50), RateOfChange(1))
        self.RegisterIndicator(self.symbol, self.tdd, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tdd.IsReady:
            # The current value of self.tdd is represented by self.tdd.Current.Value
            self.Plot("TargetDownsideDeviation", "tdd", self.tdd.Current.Value)
```

The following reference table describes the `TargetDownsideDeviation` constructor:

INDICATORS

TargetDownsideDeviation() 1/2

```
TargetDownsideDeviation QuantConnect.Indicators.TargetDownsideDeviation (
int period,
*double minimumAcceptableReturn
)
```

return are treated as underperformance of 0.

Show Details

Parameters		
int	period	The sample size of the target downside deviation.
*double	minimumAcceptableReturn	*(Optional) (Optional)* Minimum acceptable return (MAR) for target downside deviation calculation. Default: 0.

Return

`TargetDownsideDeviation` - The new `TargetDownsideDeviation` indicator object.

Definition at line 45 of file Indicators/TargetDownsideDeviation.cs.

INDICATORS

TargetDownsideDeviation() 2/2

```
TargetDownsideDeviation QuantConnect.Indicators.TargetDownsideDeviation (
string name,
int period,
*double minimumAcceptableReturn
)
```

return are treated as underperformance of 0.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The sample size of the target downside deviation.
`*double`	minimumAcceptableReturn	*(Optional) (Optional)* Minimum acceptable return (MAR) for target downside deviation calculation. Default: 0.

Return

`TargetDownsideDeviation` - The new `TargetDownsideDeviation` indicator object.

Definition at line 62 of file Indicators/TargetDownsideDeviation.cs.

Visualization

The following image shows plot values of selected properties of `TargetDownsideDeviation` using the plotly library.

TDD("SPY", 50)


variable
targetdownsidedeviation
value
0.03
0.025
0.02
0.015
0.01
0.005
Apr 2020
May 2020
Jun 2020
Jul 2020
Aug 2020
Sep 2020
Oct 2020
Nov 2020
Dec 2020
Jan 2021
index

Supported Indicators

Time Profile

Introduction

This indicator represents an Indicator of the Market Profile with Time Price Opportunity (TPO) mode and its attributes

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TP Indicator

To create an automatic indicators for `TimeProfile` , call the `TP` helper method from the `QCAlgorithm` class. The `TP` method creates a `TimeProfile` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TimeProfileAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tp = self.TP(self.symbol, 3, 0.70, 0.05)

    def OnData(self, slice: Slice) -> None:
        if self.tp.IsReady:
            # The current value of self.tp is represented by self.tp.Current.Value
            self.Plot("TimeProfile", "tp", self.tp.Current.Value)
            # Plot all attributes of self.tp
            self.Plot("TimeProfile", "volumeperprice", self.tp.VolumePerPrice.Current.Value)
            self.Plot("TimeProfile", "profilehigh", self.tp.ProfileHigh.Current.Value)
            self.Plot("TimeProfile", "profilelow", self.tp.ProfileLow.Current.Value)
            self.Plot("TimeProfile", "pocprice", self.tp.POCPrice.Current.Value)
            self.Plot("TimeProfile", "pocvolume", self.tp.POCVolume.Current.Value)
            self.Plot("TimeProfile", "valueareavolume", self.tp.ValueAreaVolume.Current.Value)
            self.Plot("TimeProfile", "valueareahigh", self.tp.ValueAreaHigh.Current.Value)
            self.Plot("TimeProfile", "valuearealow", self.tp.ValueAreaLow.Current.Value)
```

The following reference table describes the `TP` method:

INDICATORS

TP() 1/1

```
TimeProfile QuantConnect.Algorithm.QCAlgorithm.TP (
Symbol symbol,
*Int32 period,
*Decimal valueAreaVolumePercentage,
*Decimal priceRangeRoundOff,
*Resolution resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an Market Profile indicator for the symbol with Time Price Opportunity (TPO) mode. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TP we want.
`*Int32`	period	*(Optional)* The period of the TP.
`*Decimal`	valueAreaVolumePercentage	*(Optional)* The percentage of volume contained in the value area.
`*Decimal`	priceRangeRoundOff	*(Optional)* How many digits you want to round and the precision. i.e 0.01 round to two digits exactly.
`*Resolution`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`TimeProfile` - The Time Profile indicator for the given parameters.

Definition at line 998 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TimeProfile` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class TimeProfileAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tp = TimeProfile("", 3, 0.70, 0.05)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tp.Update(bar)

        if self.tp.IsReady:
            # The current value of self.tp is represented by self.tp.Current.Value
            self.Plot("TimeProfile", "tp", self.tp.Current.Value)
            # Plot all attributes of self.tp
            self.Plot("TimeProfile", "volumeperprice", self.tp.VolumePerPrice.Current.Value)
            self.Plot("TimeProfile", "profilehigh", self.tp.ProfileHigh.Current.Value)
            self.Plot("TimeProfile", "profilelow", self.tp.ProfileLow.Current.Value)
            self.Plot("TimeProfile", "pocprice", self.tp.POCPrice.Current.Value)
            self.Plot("TimeProfile", "pocvolume", self.tp.POCVolume.Current.Value)
            self.Plot("TimeProfile", "valueareavolume", self.tp.ValueAreaVolume.Current.Value)
            self.Plot("TimeProfile", "valueareahigh", self.tp.ValueAreaHigh.Current.Value)
            self.Plot("TimeProfile", "valuearealow", self.tp.ValueAreaLow.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class TimeProfileAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tp = TimeProfile("", 3, 0.70, 0.05)
        self.RegisterIndicator(self.symbol, self.tp, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tp.IsReady:
            # The current value of self.tp is represented by self.tp.Current.Value
            self.Plot("TimeProfile", "tp", self.tp.Current.Value)
            # Plot all attributes of self.tp
            self.Plot("TimeProfile", "volumeperprice", self.tp.VolumePerPrice.Current.Value)
            self.Plot("TimeProfile", "profilehigh", self.tp.ProfileHigh.Current.Value)
            self.Plot("TimeProfile", "profilelow", self.tp.ProfileLow.Current.Value)
            self.Plot("TimeProfile", "pocprice", self.tp.POCPrice.Current.Value)
            self.Plot("TimeProfile", "pocvolume", self.tp.POCVolume.Current.Value)
            self.Plot("TimeProfile", "valueareavolume", self.tp.ValueAreaVolume.Current.Value)
            self.Plot("TimeProfile", "valueareahigh", self.tp.ValueAreaHigh.Current.Value)
            self.Plot("TimeProfile", "valuearealow", self.tp.ValueAreaLow.Current.Value)
```

The following reference table describes the `TimeProfile` constructor:

INDICATORS

TimeProfile() 1/2

```
TimeProfile QuantConnect.Indicators.TimeProfile (
*int period
)
```

Creates a new TimeProfile indicator with the specified period.

Show Details

Parameters		
*int	period	*(Optional) (Optional)* The period of this indicator. Default: 2.

Return

`TimeProfile` - The new `TimeProfile` indicator object.

Definition at line 29 of file Indicators/TimeProfile.cs.

INDICATORS

TimeProfile() 2/2

```
TimeProfile QuantConnect.Indicators.TimeProfile (
string name,
int period,
*decimal valueAreaVolumePercentage
)
```

Creates a new TimeProfile indicator with the specified name, period and priceRangeRoundOff.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of this indicator.
*decimal	valueAreaVolumePercentage	*(Optional) (Optional)* The percentage of volume contained in the value area. Default: 0.70m.

Return

`TimeProfile` - The new `TimeProfile` indicator object.

Definition at line 42 of file Indicators/TimeProfile.cs.

Visualization

The following image shows plot values of selected properties of `TimeProfile` using the plotly library.

TP("SPY", 3, 0.70, 0.05)

Supported Indicators

Triangular Moving Average

Introduction

This indicator computes the Triangular Moving Average (TRIMA). The Triangular Moving Average is calculated with the following formula: (1) When the period is even, TRIMA(x,period)=SMA(SMA(x,period/2),(period/2)+1) (2) When the period is odd, TRIMA(x,period)=SMA(SMA(x,(period+1)/2),(period+1)/2)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TRIMA Indicator

To create an automatic indicators for `TriangularMovingAverage` , call the `TRIMA` helper method from the `QCAlgorithm` class. The `TRIMA` method creates a `TriangularMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class TriangularMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trima = self.TRIMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.trima.IsReady:
            # The current value of self.trima is represented by self.trima.Current.Value
            self.Plot("TriangularMovingAverage", "trima", self.trima.Current.Value)
```

The following reference table describes the `TRIMA` method:

INDICATORS

TRIMA() 1/1

```
TriangularMovingAverage QuantConnect.Algorithm.QCAlgorithm.TRIMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new TriangularMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TRIMA we want.
`Int32`	period	The period over which to compute the TRIMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TriangularMovingAverage` - The TriangularMovingAverage indicator for the requested symbol over the specified period.

Definition at line 1756 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TriangularMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class TriangularMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trima = TriangularMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.trima.Update(bar.EndTime, bar.Close)

        if self.trima.IsReady:
            # The current value of self.trima is represented by self.trima.Current.Value
            self.Plot("TriangularMovingAverage", "trima", self.trima.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class TriangularMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trima = TriangularMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.trima, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.trima.IsReady:
            # The current value of self.trima is represented by self.trima.Current.Value
            self.Plot("TriangularMovingAverage", "trima", self.trima.Current.Value)
```

The following reference table describes the `TriangularMovingAverage` constructor:

INDICATORS

TriangularMovingAverage() 1/2

```
TriangularMovingAverage QuantConnect.Indicators.TriangularMovingAverage (
string name,
int period
)
```

Initializes a new instance of the `TriangularMovingAverag` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the indicator.

Return

`TriangularMovingAverage` - The new `TriangularMovingAverage` indicator object.

Definition at line 35 of file Indicators/TriangularMovingAverage.cs.

INDICATORS

TriangularMovingAverage() 2/2

```
TriangularMovingAverage QuantConnect.Indicators.TriangularMovingAverage (
int period
)
```

Initializes a new instance of the `TriangularMovingAverag` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the indicator.

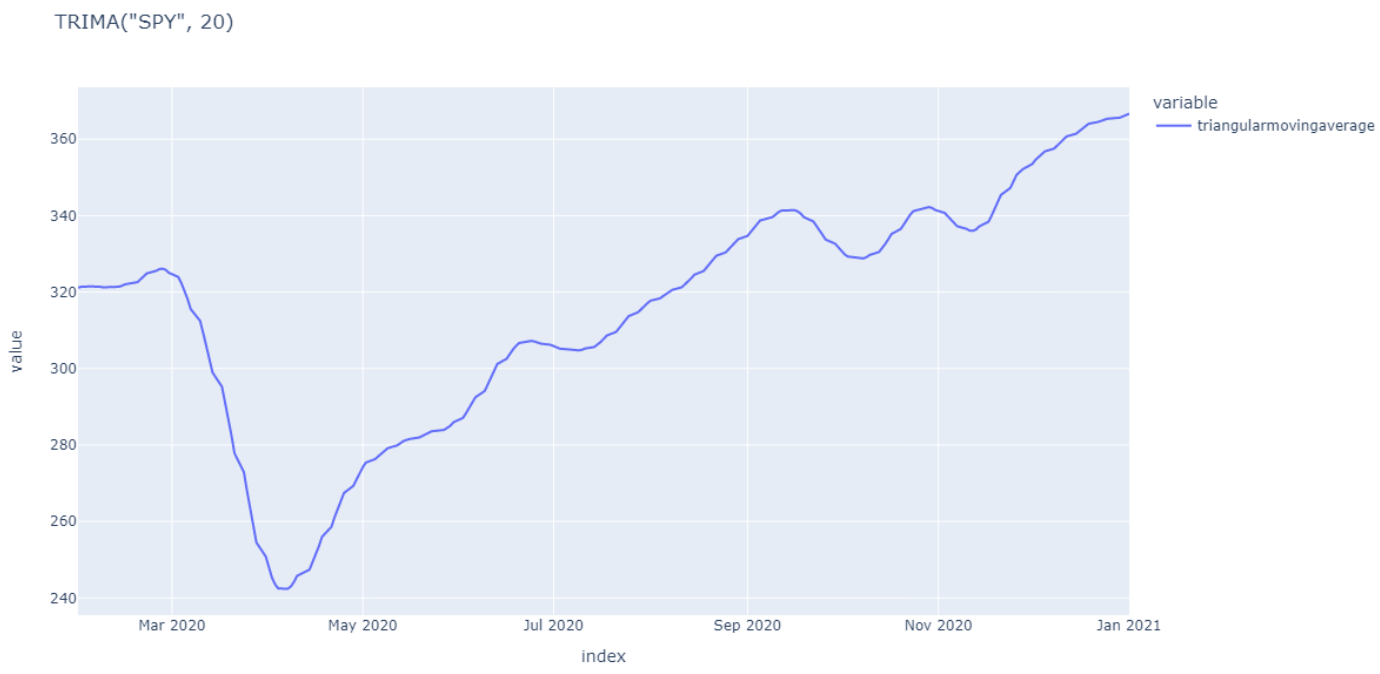
Return

`TriangularMovingAverage` - The new `TriangularMovingAverage` indicator object.

Definition at line 51 of file Indicators/TriangularMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `TriangularMovingAverage` using the plotly library.



Supported Indicators

Triple Exponential Moving Average

Introduction

This indicator computes the Triple Exponential Moving Average (TEMA). The Triple Exponential Moving Average is calculated with the following formula: EMA1 = EMA(t,period) EMA2 = EMA(EMA(t,period),period) EMA3 = EMA(EMA(EMA(t,period),period),period) TEMA = 3 * EMA1 - 3 * EMA2 + EMA3

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TEMA Indicator

To create an automatic indicators for `TripleExponentialMovingAverage` , call the `TEMA` helper method from the `QCAlgorithm` class. The `TEMA` method creates a `TripleExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TripleExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tema = self.TEMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.tema.IsReady:
            # The current value of self.tema is represented by self.tema.Current.Value
            self.Plot("TripleExponentialMovingAverage", "tema", self.tema.Current.Value)
```

The following reference table describes the `TEMA` method:

INDICATORS

TEMA() 1/1

```
TripleExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.TEMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new TripleExponentialMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TEMA we want.
`Int32`	period	The period over which to compute the TEMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TripleExponentialMovingAverage` - The TripleExponentialMovingAverage indicator for the requested symbol over the specified period.

Definition at line 1698 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TripleExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class TripleExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tema = TripleExponentialMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tema.Update(bar.EndTime, bar.Close)

        if self.tema.IsReady:
            # The current value of self.tema is represented by self.tema.Current.Value
            self.Plot("TripleExponentialMovingAverage", "tema", self.tema.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```
class TripleExponentialMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tema = TripleExponentialMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.tema, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tema.IsReady:
            # The current value of self.tema is represented by self.tema.Current.Value
            self.Plot("TripleExponentialMovingAverage", "tema", self.tema.Current.Value)
```

The following reference table describes the `TripleExponentialMovingAverage` constructor:

INDICATORS

TripleExponentialMovingAverage() 1/2

```
TripleExponentialMovingAverage QuantConnect.Indicators.TripleExponentialMovingAverage (
string name,
int period
)
```

Initializes a new instance of the `TripleExponentialMovingAverag` class using the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the TEMA.

Return

`TripleExponentialMovingAverage` - The new `TripleExponentialMovingAverage` indicator object.

Definition at line 38 of file Indicators/TripleExponentialMovingAverage.cs.

INDICATORS

TripleExponentialMovingAverage() 2/2

```
TripleExponentialMovingAverage QuantConnect.Indicators.TripleExponentialMovingAverage (
int period
)
```

Initializes a new instance of the `TripleExponentialMovingAverag` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the TEMA.

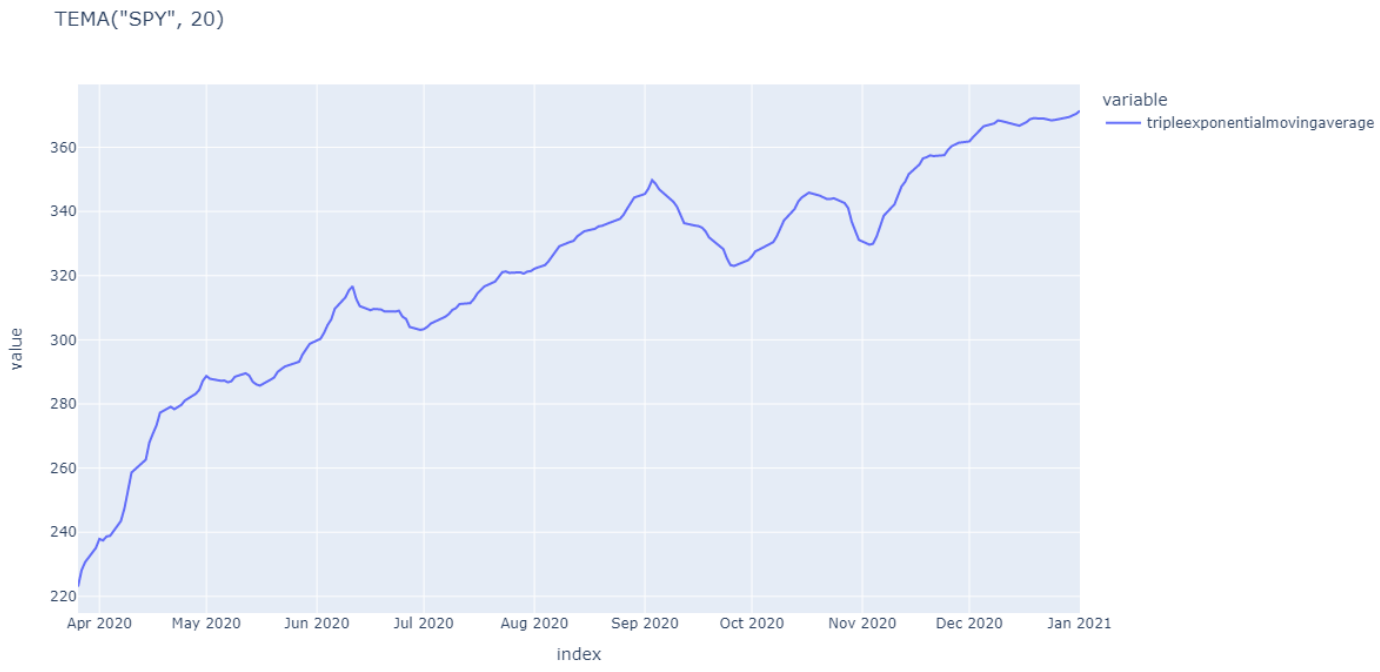
Return

`TripleExponentialMovingAverage` - The new `TripleExponentialMovingAverage` indicator object.

Definition at line 51 of file Indicators/TripleExponentialMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `TripleExponentialMovingAverage` using the plotly library.



Supported Indicators

Trix

Introduction

This indicator computes the TRIX (1-period ROC of a Triple EMA) The TRIX is calculated as explained here: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:trix

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TRIX Indicator

To create an automatic indicators for `Trix` , call the `TRIX` helper method from the `QCAlgorithm` class. The `TRIX` method creates a `Trix` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
class TrixAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trix = self.TRIX(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.trix.IsReady:
            # The current value of self.trix is represented by self.trix.Current.Value
            self.Plot("Trix", "trix", self.trix.Current.Value)
```

The following reference table describes the `TRIX` method:

INDICATORS

TRIX() 1/1

```
Trix QuantConnect.Algorithm.QCAlgorithm.TRIX (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Trix indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TRIX we want.
`Int32`	period	The period over which to compute the TRIX.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Trix` - The Trix indicator for the requested symbol over the specified period.

Definition at line 1774 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Trix` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class TrixAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trix = Trix(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.trix.Update(bar.EndTime, bar.Close)

        if self.trix.IsReady:
            # The current value of self.trix is represented by self.trix.Current.Value
            self.Plot("Trix", "trix", self.trix.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class TrixAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.trix = Trix(20)
        self.RegisterIndicator(self.symbol, self.trix, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.trix.IsReady:
            # The current value of self.trix is represented by self.trix.Current.Value
            self.Plot("Trix", "trix", self.trix.Current.Value)
```

The following reference table describes the `Trix` constructor:

INDICATORS

Trix() 1/2

```
Trix QuantConnect.Indicators.Trix (
string name,
int period
)
```

Initializes a new instance of the `Tri` class using the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the indicator.

Return

`Trix` - The new `Trix` indicator object.

Definition at line 36 of file Indicators/Trix.cs.

INDICATORS

Trix() 2/2

```
Trix QuantConnect.Indicators.Trix (
int period
)
```

Initializes a new instance of the `Tri` class using the specified period.

Show Details

Parameters		
`int`	period	The period of the indicator.

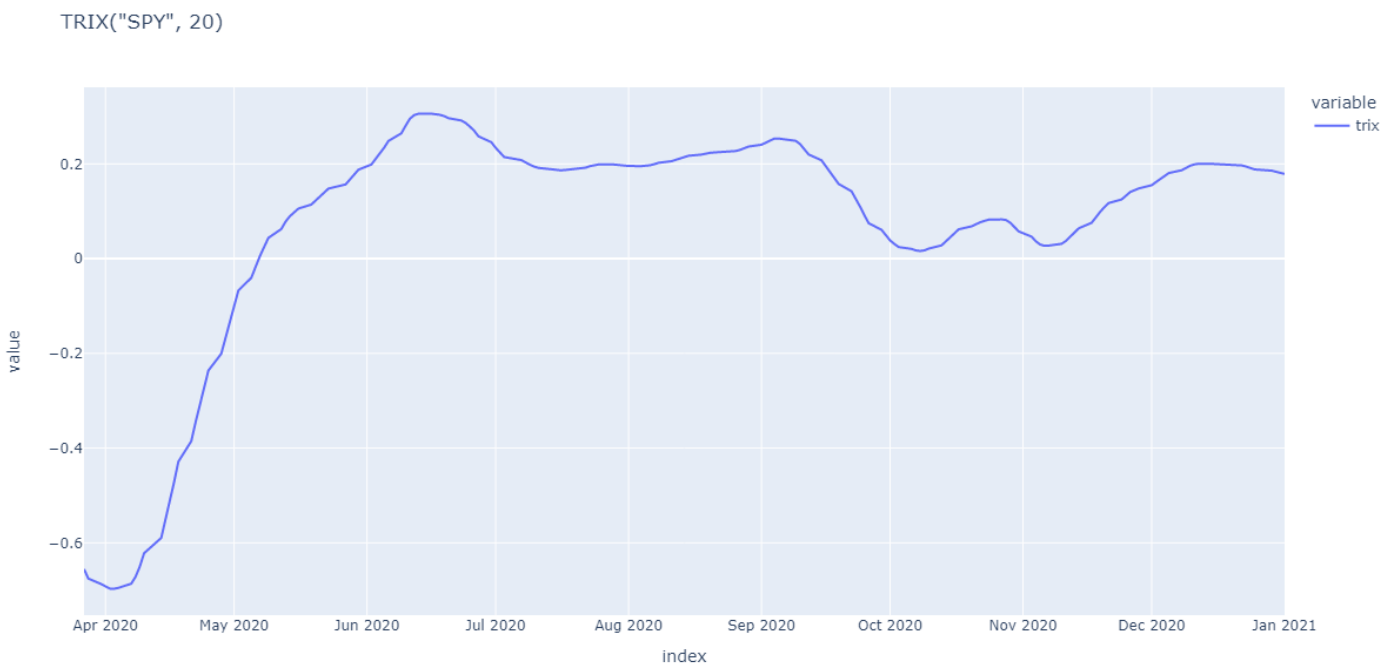
Return

`Trix` - The new `Trix` indicator object.

Definition at line 50 of file Indicators/Trix.cs.

Visualization

The following image shows plot values of selected properties of `Trix` using the plotly library.



Supported Indicators

True Range

Introduction

This indicator computes the True Range (TR). The True Range is the greatest of the following values: value1 = distance from today's high to today's low. value2 = distance from yesterday's close to today's high. value3 = distance from yesterday's close to today's low.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TR Indicator

To create an automatic indicators for `TrueRange` , call the `TR` helper method from the `QCAlgorithm` class. The `TR` method creates a `TrueRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class TrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tr = self.TR(self.symbol)

    def OnData(self, slice: Slice) -> None:
        if self.tr.IsReady:
            # The current value of self.tr is represented by self.tr.Current.Value
            self.Plot("TrueRange", "tr", self.tr.Current.Value)
```

The following reference table describes the `TR` method:

INDICATORS

TR() 1/1

```
TrueRange QuantConnect.Algorithm.QCAlgorithm.TR (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new TrueRange indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TR we want.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`TrueRange` - The TrueRange indicator for the requested symbol.

Definition at line 1738 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `TrueRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class TrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tr = TrueRange()

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tr.Update(bar)

        if self.tr.IsReady:
            # The current value of self.tr is represented by self.tr.Current.Value
            self.Plot("TrueRange", "tr", self.tr.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class TrueRangeAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tr = TrueRange()
        self.RegisterIndicator(self.symbol, self.tr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tr.IsReady:
            # The current value of self.tr is represented by self.tr.Current.Value
            self.Plot("TrueRange", "tr", self.tr.Current.Value)
```

The following reference table describes the `TrueRange` constructor:

INDICATORS

TrueRange() 1/2

```
TrueRange QuantConnect.Indicators.TrueRange (

)
```

Initializes a new instance of the `TrueRang` class using the specified name.

Show Details

This method requires no argument input.

Return

`TrueRange` - The new `TrueRange` indicator object.

Definition at line 35 of file Indicators/TrueRange.cs.

INDICATORS

TrueRange() 2/2

```
TrueRange QuantConnect.Indicators.TrueRange (
string name
)
```

Initializes a new instance of the `TrueRang` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.

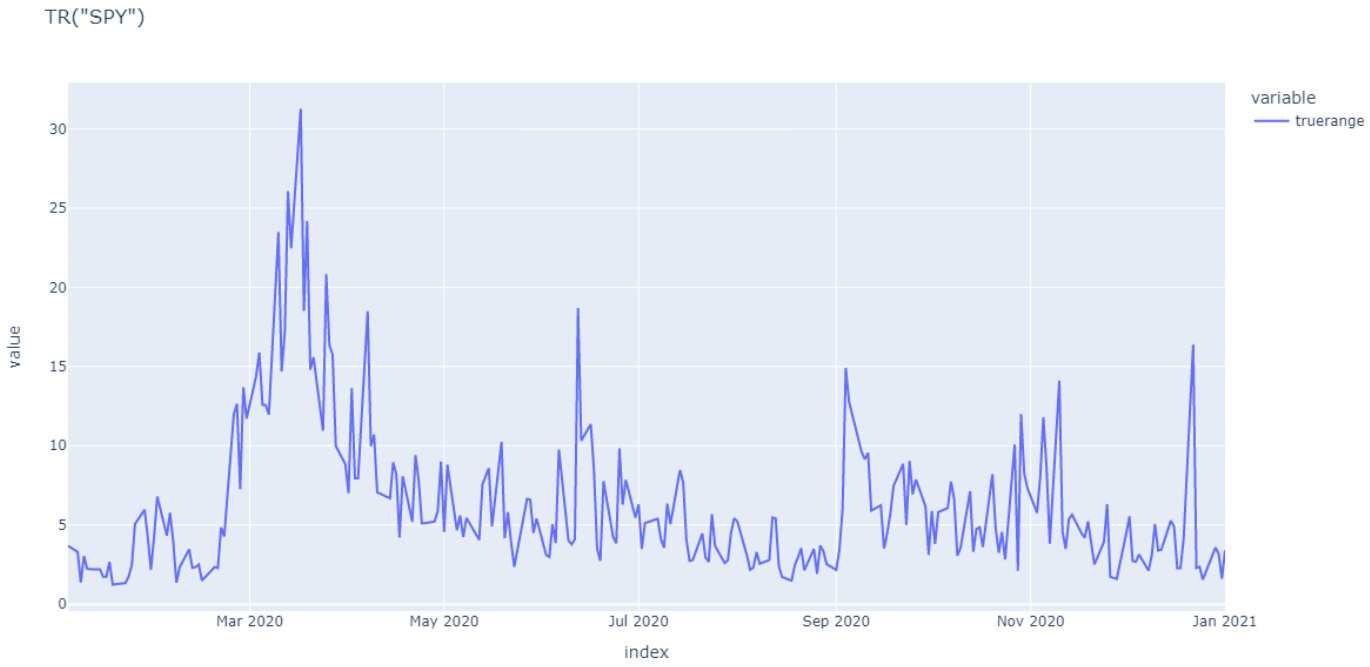
Return

`TrueRange` - The new `TrueRange` indicator object.

Definition at line 44 of file Indicators/TrueRange.cs.

Visualization

The following image shows plot values of selected properties of `TrueRange` using the plotly library.



Supported Indicators

True Strength Index

Introduction

This indicator computes the True Strength Index (TSI). The True Strength Index is calculated as explained here: https://school.stockcharts.com/doku.php?id=technical_indicators:True_strength_index Briefly, the calculation has three steps: 1. Smooth the momentum and the absolute momentum by getting an EMA of them (typically of period 25) 2. Double smooth the momentum and the absolute momentum by getting an EMA of their EMA (typically of period 13) 3. The TSI formula itself: divide the double-smoothed momentum over the double-smoothed absolute momentum and multiply by 100 The signal is typically a 7-to-12-EMA of the TSI.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TSI Indicator

To create an automatic indicators for `TrueStrengthIndex` , call the `TSI` helper method from the `QCAlgorithm` class. The `TSI` method creates a `TrueStrengthIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class TrueStrengthIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tsi = self.TSI(self.symbol, 25, 13, 7, MovingAverageType.Exponential)

    def OnData(self, slice: Slice) -> None:
        if self.tsi.IsReady:
            # The current value of self.tsi is represented by self.tsi.Current.Value
            self.Plot("TrueStrengthIndex", "tsi", self.tsi.Current.Value)
            # Plot all attributes of self.tsi
            self.Plot("TrueStrengthIndex", "signal", self.tsi.Signal.Current.Value)
```

The following reference table describes the `TSI` method:

INDICATORS

TSI() 1/1

```
TrueStrengthIndex QuantConnect.Algorithm.QCAlgorithm.TSI (
Symbol symbol,
*Int32 longTermPeriod,
*Int32 shortTermPeriod,
*Int32 signalPeriod,
*MovingAverageType signalType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a TrueStrengthIndex indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TSI we want.
`*Int32`	longTermPeriod	*(Optional)* Period used for the second (double) price change smoothing.
`*Int32`	shortTermPeriod	*(Optional)* Period used for the first price change smoothing.
`*Int32`	signalPeriod	*(Optional)* The signal period.
`*MovingAverageType`	signalType	*(Optional)* The type of moving average to use for the signal.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TrueStrengthIndex` - The TrueStrengthIndex indicator for the given parameters.

Definition at line 1720 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

The following table describes the `MovingAverageType` enumeration members:

You can manually create a `TrueStrengthIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

```py
class TrueStrengthIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tsi = TrueStrengthIndex(25, 13, 7, MovingAverageType.Exponential)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.tsi.Update(bar.EndTime, bar.Close)

        if self.tsi.IsReady:
            # The current value of self.tsi is represented by self.tsi.Current.Value
            self.Plot("TrueStrengthIndex", "tsi", self.tsi.Current.Value)
            # Plot all attributes of self.tsi
            self.Plot("TrueStrengthIndex", "signal", self.tsi.Signal.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class TrueStrengthIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.tsi = TrueStrengthIndex(25, 13, 7, MovingAverageType.Exponential)
        self.RegisterIndicator(self.symbol, self.tsi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.tsi.IsReady:
            # The current value of self.tsi is represented by self.tsi.Current.Value
            self.Plot("TrueStrengthIndex", "tsi", self.tsi.Current.Value)
            # Plot all attributes of self.tsi
            self.Plot("TrueStrengthIndex", "signal", self.tsi.Signal.Current.Value)
```

The following reference table describes the `TrueStrengthIndex` constructor:

INDICATORS

TrueStrengthIndex() 1/2

```
TrueStrengthIndex QuantConnect.Indicators.TrueStrengthIndex (
*int shortTermPeriod,
*int longTermPeriod,
*int signalPeriod,
*MovingAverageType signalType
)
```

Initializes a new instance of the `TrueStrengthInde` class using the specified short and long term smoothing periods, and the signal period and type.

Show Details

Parameters		
*int	shortTermPeriod	*(Optional) (Optional)* Period used for the first price change smoothing. Default: 13.
*int	longTermPeriod	*(Optional) (Optional)* Period used for the second (double) price change smoothing. Default: 25.
*int	signalPeriod	*(Optional) (Optional)* The signal period. Default: 7.
*MovingAverageType	signalType	*(Optional) (Optional)* The type of moving average to use for the signal. Default: MovingAverageType.Exponential.

Return

`TrueStrengthIndex` - The new `TrueStrengthIndex` indicator object.

Definition at line 67 of file Indicators/TrueStrengthIndex.cs.

INDICATORS

TrueStrengthIndex() 2/2

```
TrueStrengthIndex QuantConnect.Indicators.TrueStrengthIndex (
string name,
*int shortTermPeriod,
*int longTermPeriod,
*int signalPeriod,
*MovingAverageType signalType
)
```

Initializes a new instance of the `TrueStrengthInde` class using the specified name, the short and long term smoothing periods, and the signal period and type.

Show Details

Parameters		
`string`	name	The name of the indicator.
`*int`	shortTermPeriod	*(Optional) (Optional)* Period used for the first price change smoothing. Default: 13.
`*int`	longTermPeriod	*(Optional) (Optional)* Period used for the second (double) price change smoothing. Default: 25.
`*int`	signalPeriod	*(Optional) (Optional)* The signal period. Default: 7.
`*MovingAverageType`	signalType	*(Optional) (Optional)* The type of moving average to use for the signal. Default: MovingAverageType.Exponential.

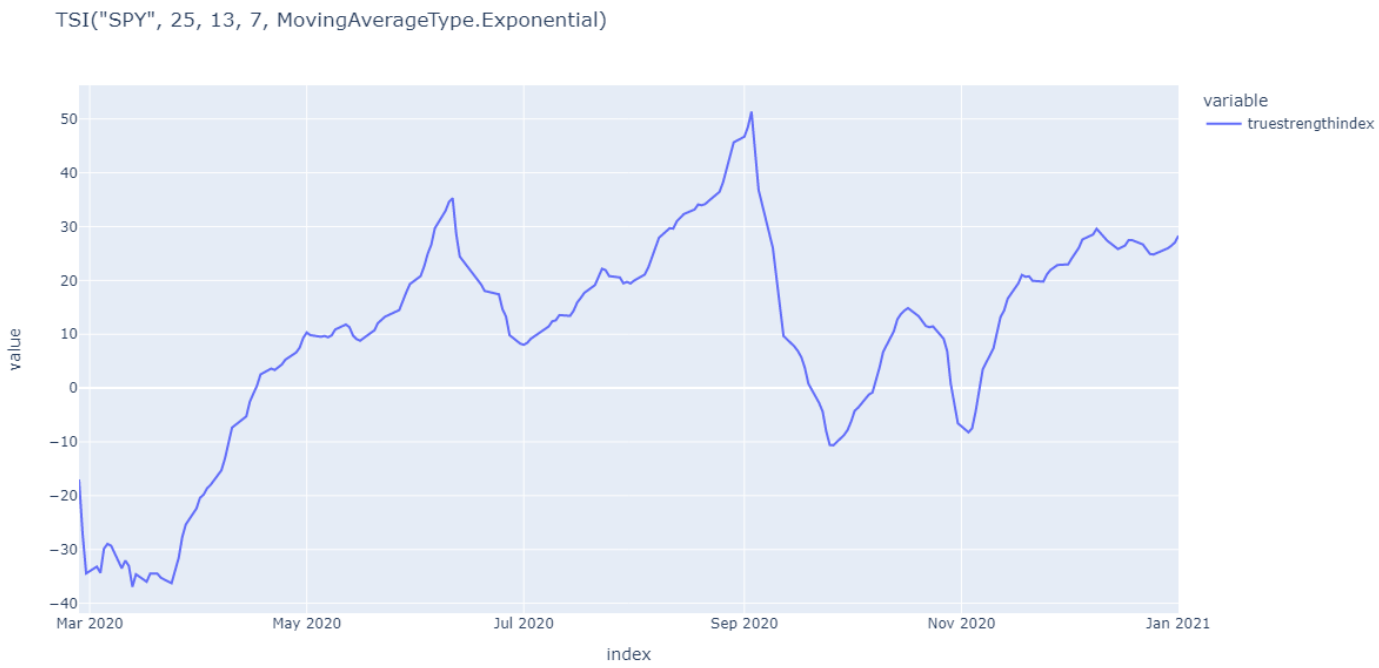
Return

`TrueStrengthIndex` - The new `TrueStrengthIndex` indicator object.

Definition at line 80 of file Indicators/TrueStrengthIndex.cs.

Visualization

The following image shows plot values of selected properties of `TrueStrengthIndex` using the plotly library.



Supported Indicators

Ultimate Oscillator

Introduction

This indicator computes the Ultimate Oscillator (ULTOSC) The Ultimate Oscillator is calculated as explained here: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:ultimate_oscillator

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ULTOSC Indicator

To create an automatic indicators for `UltimateOscillator` , call the `ULTOSC` helper method from the `QCAlgorithm` class. The `ULTOSC` method creates a `UltimateOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
PY
class UltimateOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ultosc = self.ULTOSC(self.symbol, 5, 10, 20)

    def OnData(self, slice: Slice) -> None:
        if self.ultosc.IsReady:
            # The current value of self.ultosc is represented by self.ultosc.Current.Value
            self.Plot("UltimateOscillator", "ultosc", self.ultosc.Current.Value)
```

The following reference table describes the `ULTOSC` method:

INDICATORS

ULTOSC() 1/1

```
UltimateOscillator QuantConnect.Algorithm.QCAlgorithm.ULTOSC (
Symbol symbol,
Int32 period1,
Int32 period2,
Int32 period3,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new UltimateOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ULTOSC we want.
`Int32`	period1	The first period over which to compute the ULTOSC.
`Int32`	period2	The second period over which to compute the ULTOSC.
`Int32`	period3	The third period over which to compute the ULTOSC.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`UltimateOscillator` - The UltimateOscillator indicator for the requested symbol over the specified period.

Definition at line 1794 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `UltimateOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class UltimateOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ultosc = UltimateOscillator(5, 10, 20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.ultosc.Update(bar)

        if self.ultosc.IsReady:
            # The current value of self.ultosc is represented by self.ultosc.Current.Value
            self.Plot("UltimateOscillator", "ultosc", self.ultosc.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class UltimateOscillatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.ultosc = UltimateOscillator(5, 10, 20)
        self.RegisterIndicator(self.symbol, self.ultosc, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.ultosc.IsReady:
            # The current value of self.ultosc is represented by self.ultosc.Current.Value
            self.Plot("UltimateOscillator", "ultosc", self.ultosc.Current.Value)
```

The following reference table describes the `UltimateOscillator` constructor:

INDICATORS

UltimateOscillator() 1/2

```
UltimateOscillator QuantConnect.Indicators.UltimateOscillator (
int period1,
int period2,
int period3
)
```

Initializes a new instance of the `UltimateOscillato` class using the specified parameters.

Show Details

Parameters		
int	period1	The first period.
int	period2	The second period.
int	period3	The third period.

Return

`UltimateOscillator` - The new `UltimateOscillator` indicator object.

Definition at line 44 of file Indicators/UltimateOscillator.cs.

INDICATORS

UltimateOscillator() 2/2

```
UltimateOscillator QuantConnect.Indicators.UltimateOscillator (
 string name,
 int period1,
 int period2,
 int period3
 )
```

Initializes a new instance of the `UltimateOscillato` class using the specified parameters.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period1	The first period.
`int`	period2	The second period.
`int`	period3	The third period.

Return

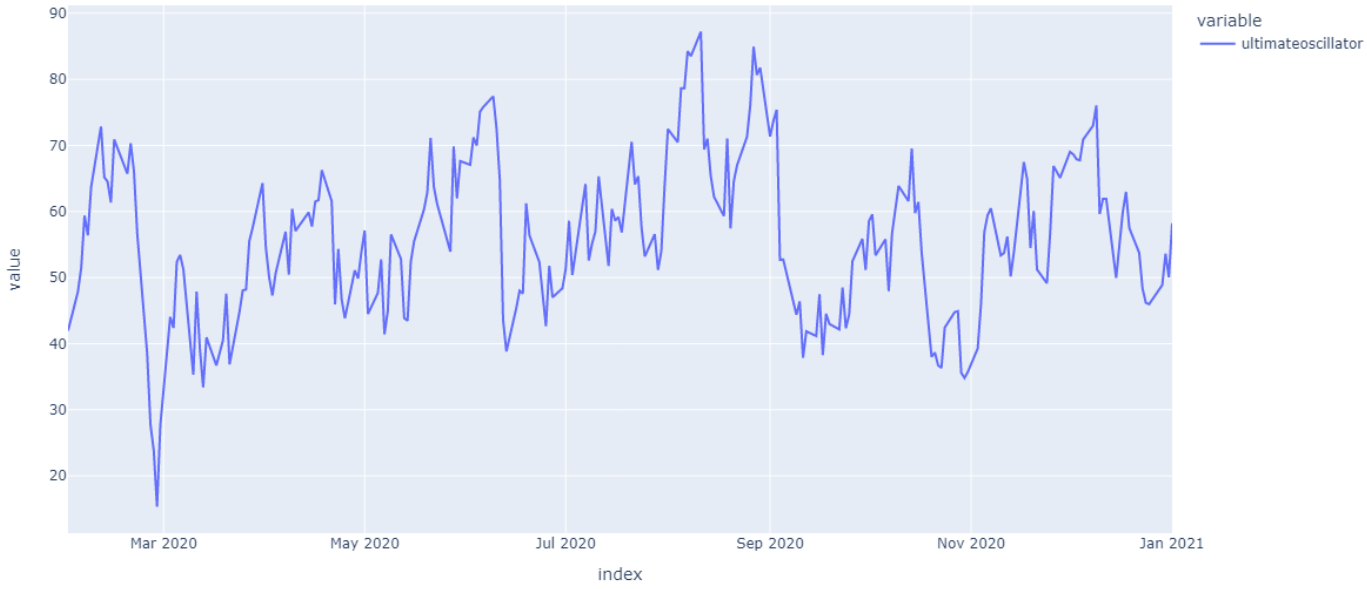
`UltimateOscillator` - The new `UltimateOscillator` indicator object.

Definition at line 56 of file Indicators/UltimateOscillator.cs.

Visualization

The following image shows plot values of selected properties of `UltimateOscillator` using the plotly library.

ULTOSC("SPY", 5, 10, 20)



Supported Indicators

Variance

Introduction

This indicator computes the n-period population variance.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VAR Indicator

To create an automatic indicators for `Variance` , call the `VAR` helper method from the `QCAlgorithm` class. The `VAR` method creates a `Variance` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class VarianceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.var = self.VAR(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.var.IsReady:
            # The current value of self.var is represented by self.var.Current.Value
            self.Plot("Variance", "var", self.var.Current.Value)
```

The following reference table describes the `VAR` method:

INDICATORS

VAR() 1/1

```
Variance QuantConnect.Algorithm.QCAlgorithm.VAR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Variance indicator. This will return the population variance of samples over the specified period.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VAR we want.
`Int32`	period	The period over which to compute the VAR.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Variance` - The Variance indicator for the requested symbol over the specified period.

Definition at line 1812 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `Variance` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class VarianceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.var = Variance(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.var.Update(bar.EndTime, bar.Close)

        if self.var.IsReady:
            # The current value of self.var is represented by self.var.Current.Value
            self.Plot("Variance", "var", self.var.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class VarianceAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.var = Variance(20)
        self.RegisterIndicator(self.symbol, self.var, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.var.IsReady:
            # The current value of self.var is represented by self.var.Current.Value
            self.Plot("Variance", "var", self.var.Current.Value)
```

The following reference table describes the `Variance` constructor:

INDICATORS

Variance() 1/2

```
Variance QuantConnect.Indicators.Variance (
int period
)
```

Initializes a new instance of the `Varianc` class using the specified period.

Show Details

Parameters		
int	period	The period of the indicator.

Return

`Variance` - The new `Variance` indicator object.

Definition at line 32 of file Indicators/Variance.cs.

INDICATORS

Variance() 2/2

```
Variance QuantConnect.Indicators.Variance (
string name,
int period
)
```

Initializes a new instance of the `Varianc` class using the specified name and period.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The period of the indicator.

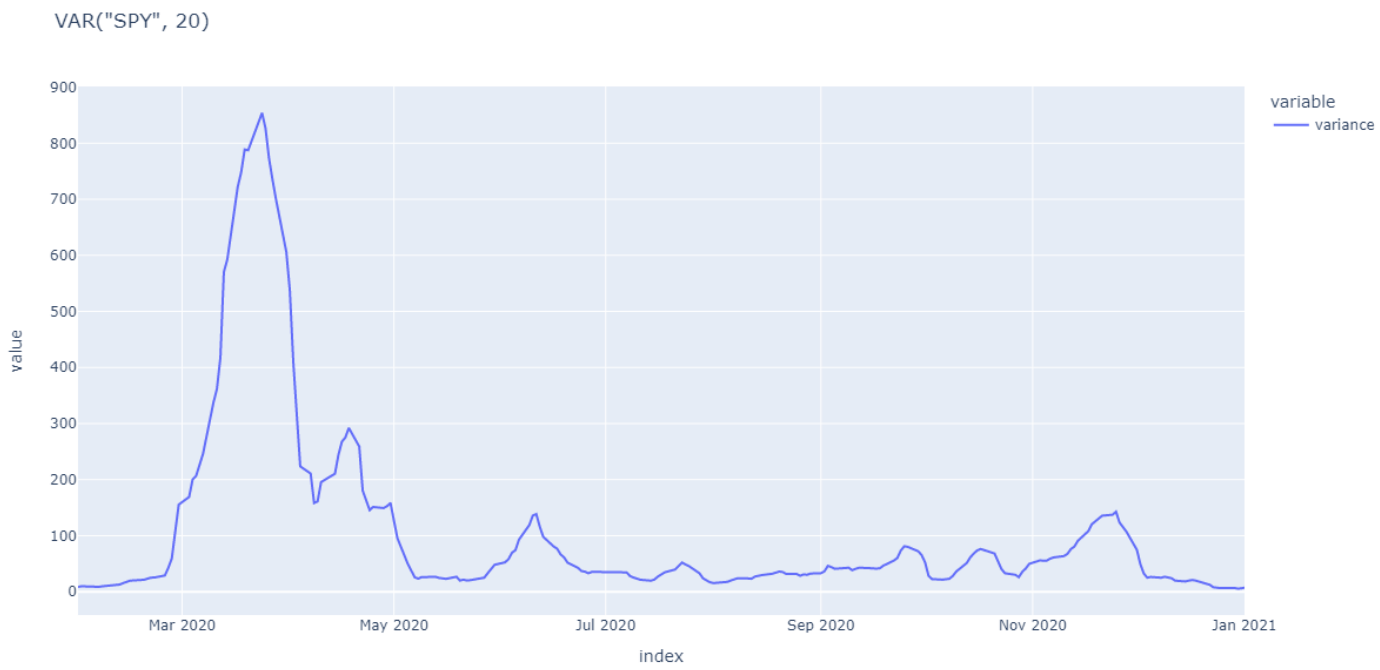
Return

`Variance` - The new `Variance` indicator object.

Definition at line 42 of file Indicators/Variance.cs.

Visualization

The following image shows plot values of selected properties of `Variance` using the plotly library.



Supported Indicators

Volume Profile

Introduction

This indicator represents an Indicator of the Market Profile with Volume Profile mode and its attributes

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VP Indicator

To create an automatic indicators for `VolumeProfile` , call the `VP` helper method from the `QCAlgorithm` class. The `VP` method creates a `VolumeProfile` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class VolumeProfileAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vp = self.VP(self.symbol, 3, 0.70, 0.05)

    def OnData(self, slice: Slice) -> None:
        if self.vp.IsReady:
            # The current value of self.vp is represented by self.vp.Current.Value
            self.Plot("VolumeProfile", "vp", self.vp.Current.Value)
            # Plot all attributes of self.vp
            self.Plot("VolumeProfile", "volumeperprice", self.vp.VolumePerPrice.Current.Value)
            self.Plot("VolumeProfile", "profilehigh", self.vp.ProfileHigh.Current.Value)
            self.Plot("VolumeProfile", "profilelow", self.vp.ProfileLow.Current.Value)
            self.Plot("VolumeProfile", "pocprice", self.vp.POCPrice.Current.Value)
            self.Plot("VolumeProfile", "pocvolume", self.vp.POCVolume.Current.Value)
            self.Plot("VolumeProfile", "valueareavolume", self.vp.ValueAreaVolume.Current.Value)
            self.Plot("VolumeProfile", "valueareahigh", self.vp.ValueAreaHigh.Current.Value)
            self.Plot("VolumeProfile", "valuearealow", self.vp.ValueAreaLow.Current.Value)
```

The following reference table describes the `VP` method:

INDICATORS

VP() 1/1

```
VolumeProfile QuantConnect.Algorithm.QCAlgorithm.VP (
Symbol symbol,
*Int32 period,
*Decimal valueAreaVolumePercentage,
*Decimal priceRangeRoundOff,
*Resolution resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an Market Profile indicator for the symbol with Volume Profile (VOL) mode. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VP we want.
`*Int32`	period	*(Optional)* The period of the VP.
`*Decimal`	valueAreaVolumePercentage	*(Optional)* The percentage of volume contained in the value area.
`*Decimal`	priceRangeRoundOff	*(Optional)* How many digits you want to round and the precision. i.e 0.01 round to two digits exactly.
`*Resolution`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`VolumeProfile` - The Volume Profile indicator for the given parameters.

Definition at line 977 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `VolumeProfile` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class VolumeProfileAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vp = VolumeProfile("", 3, 0.70, 0.05)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.vp.Update(bar)

        if self.vp.IsReady:
            # The current value of self.vp is represented by self.vp.Current.Value
            self.Plot("VolumeProfile", "vp", self.vp.Current.Value)
            # Plot all attributes of self.vp
            self.Plot("VolumeProfile", "volumeperprice", self.vp.VolumePerPrice.Current.Value)
            self.Plot("VolumeProfile", "profilehigh", self.vp.ProfileHigh.Current.Value)
            self.Plot("VolumeProfile", "profilelow", self.vp.ProfileLow.Current.Value)
            self.Plot("VolumeProfile", "pocprice", self.vp.POCPrice.Current.Value)
            self.Plot("VolumeProfile", "pocvolume", self.vp.POCVolume.Current.Value)
            self.Plot("VolumeProfile", "valueareavolume", self.vp.ValueAreaVolume.Current.Value)
            self.Plot("VolumeProfile", "valueareahigh", self.vp.ValueAreaHigh.Current.Value)
            self.Plot("VolumeProfile", "valuearealow", self.vp.ValueAreaLow.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class VolumeProfileAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vp = VolumeProfile("", 3, 0.70, 0.05)
        self.RegisterIndicator(self.symbol, self.vp, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.vp.IsReady:
            # The current value of self.vp is represented by self.vp.Current.Value
            self.Plot("VolumeProfile", "vp", self.vp.Current.Value)
            # Plot all attributes of self.vp
            self.Plot("VolumeProfile", "volumeperprice", self.vp.VolumePerPrice.Current.Value)
            self.Plot("VolumeProfile", "profilehigh", self.vp.ProfileHigh.Current.Value)
            self.Plot("VolumeProfile", "profilelow", self.vp.ProfileLow.Current.Value)
            self.Plot("VolumeProfile", "pocprice", self.vp.POCPrice.Current.Value)
            self.Plot("VolumeProfile", "pocvolume", self.vp.POCVolume.Current.Value)
            self.Plot("VolumeProfile", "valueareavolume", self.vp.ValueAreaVolume.Current.Value)
            self.Plot("VolumeProfile", "valueareahigh", self.vp.ValueAreaHigh.Current.Value)
            self.Plot("VolumeProfile", "valuearealow", self.vp.ValueAreaLow.Current.Value)
```

The following reference table describes the `VolumeProfile` constructor:

INDICATORS

VolumeProfile() 1/2

```
VolumeProfile QuantConnect.Indicators.VolumeProfile (
*int period
)
```

Creates a new VolumeProfile indicator with the specified period.

Show Details

Parameters		
*int	period	*(Optional) (Optional)* The period of the indicator. Default: 2.

Return

VolumeProfile - The new VolumeProfile indicator object.

Definition at line 29 of file Indicators/VolumeProfile.cs.

INDICATORS

VolumeProfile() 2/2

```
VolumeProfile QuantConnect.Indicators.VolumeProfile (
string name,
int period,
*decimal valueAreaVolumePercentage
)
```

Creates a new VolumeProfile indicator with the specified name, period and priceRangeRoundOff.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of this indicator.
*decimal	valueAreaVolumePercentage	*(Optional) (Optional)* The percentage of volume contained in the value area. Default: 0.70m.

Return

VolumeProfile - The new VolumeProfile indicator object.

Definition at line 42 of file Indicators/VolumeProfile.cs.

Visualization

The following image shows plot values of selected properties of VolumeProfile using the plotly library.


VP("SPY", 3, 0.70, 0.05)
variable
volumeprofile
value
index
380
360
340
320
300
280
260
240
220
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021

Supported Indicators

Volume Weighted Average Price Indicator

Introduction

Volume Weighted Average Price (VWAP) Indicator: It is calculated by adding up the dollars traded for every transaction (price multiplied by number of shares traded) and then dividing by the total shares traded for the day.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VWAP Indicator

To create an automatic indicators for `VolumeWeightedAveragePriceIndicator` , call the `VWAP` helper method from the `QCAlgorithm` class. The `VWAP` method creates a `VolumeWeightedAveragePriceIndicator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class VolumeWeightedAveragePriceIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vwap = self.VWAP(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.vwap.IsReady:
            # The current value of self.vwap is represented by self.vwap.Current.Value
            self.Plot("VolumeWeightedAveragePriceIndicator", "vwap", self.vwap.Current.Value)
```

The following reference table describes the `VWAP` method:

INDICATORS

VWAP() 1/2

```
VolumeWeightedAveragePriceIndicator QuantConnect.Algorithm.QCAlgorithm.VWAP (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an VolumeWeightedAveragePrice (VWAP) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VWAP we want.
`Int32`	period	The period of the VWAP.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`VolumeWeightedAveragePriceIndicator` - The VolumeWeightedAveragePrice for the given parameters.

Definition at line 1831 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

VWAP() 2/2

```
IntradayVwap QuantConnect.Algorithm.QCAlgorithm.VWAP (
 Symbol symbol
)
```

Creates the canonical VWAP indicator that resets each day. The indicator will be automatically updated on the security's configured resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VWAP we want.

Return

`IntradayVwap` - The IntradayVWAP for the specified symbol.

Definition at line 1847 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update

its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `VolumeWeightedAveragePriceIndicator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class VolumeWeightedAveragePriceIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vwap = VolumeWeightedAveragePriceIndicator(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.vwap.Update(bar)

        if self.vwap.IsReady:
            # The current value of self.vwap is represented by self.vwap.Current.Value
            self.Plot("VolumeWeightedAveragePriceIndicator", "vwap", self.vwap.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class VolumeWeightedAveragePriceIndicatorAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.vwap = VolumeWeightedAveragePriceIndicator(20)
        self.RegisterIndicator(self.symbol, self.vwap, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.vwap.IsReady:
            # The current value of self.vwap is represented by self.vwap.Current.Value
            self.Plot("VolumeWeightedAveragePriceIndicator", "vwap", self.vwap.Current.Value)
```

The following reference table describes the `VolumeWeightedAveragePriceIndicator` constructor:

INDICATORS

VolumeWeightedAveragePriceIndicator() 1/2

```
VolumeWeightedAveragePriceIndicator QuantConnect.Indicators.VolumeWeightedAveragePriceIndicator (
int period
)
```

Initializes a new instance of the VWAP class with the default name and period.

Show Details

Parameters		
int	period	The period of the VWAP.

Return

`VolumeWeightedAveragePriceIndicator` - The new `VolumeWeightedAveragePriceIndicator` indicator object.

Definition at line 40 of file Indicators/VolumeWeightedAveragePriceIndicator.cs.

INDICATORS

VolumeWeightedAveragePriceIndicator() 2/2

```
VolumeWeightedAveragePriceIndicator QuantConnect.Indicators.VolumeWeightedAveragePriceIndicator (
string name,
int period
)
```

Initializes a new instance of the VWAP class with a given name and period.

Show Details

Parameters		
string	name	string - the name of the indicator.
int	period	The period of the VWAP.

Return

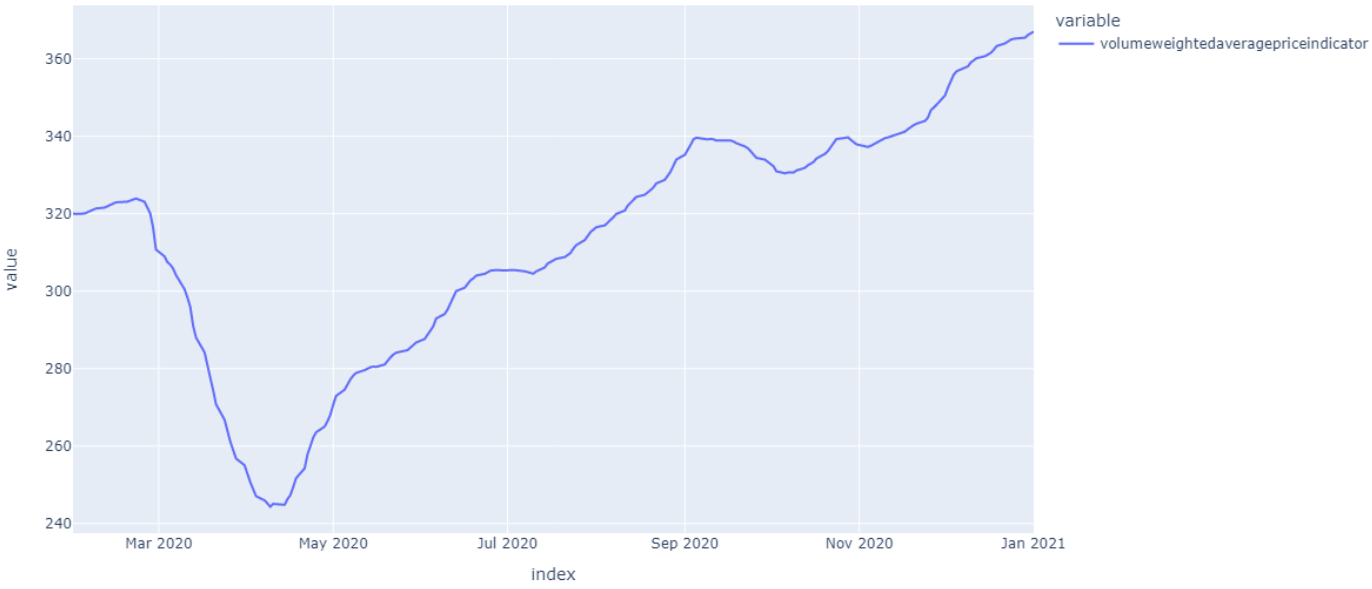
`VolumeWeightedAveragePriceIndicator` - The new `VolumeWeightedAveragePriceIndicator` indicator object.

Definition at line 50 of file Indicators/VolumeWeightedAveragePriceIndicator.cs.

Visualization

The following image shows plot values of selected properties of `VolumeWeightedAveragePriceIndicator` using the plotly library.

VWAP("SPY", 20)


variable
volumeweightedaveragepriceindicator
360
340
320
300
280
260
240
value
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Wilder Accumulative Swing Index

Introduction

This indicator calculates the Accumulative Swing Index (ASI) as defined by Welles Wilder in his book 'New Concepts in Technical Trading Systems'. $ASI_t = ASI_{t-1} + SI_t$ Where: ASI_{t-1} The for the previous period. SI_t The calculated for the current period.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ASI Indicator

To create an automatic indicators for `WilderAccumulativeSwingIndex` , call the `ASI` helper method from the `QCAlgorithm` class. The `ASI` method creates a `WilderAccumulativeSwingIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class WilderAccumulativeSwingIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.asi = self.ASI(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.asi.IsReady:
            # The current value of self.asi is represented by self.asi.Current.Value
            self.Plot("WilderAccumulativeSwingIndex", "asi", self.asi.Current.Value)
```

The following reference table describes the `ASI` method:

INDICATORS

ASI() 1/1

```
WilderAccumulativeSwingIndex QuantConnect.Algorithm.QCAlgorithm.ASI (
Symbol symbol,
Decimal limitMove,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a Wilder Accumulative Swing Index (ASI) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ASI we want.
`Decimal`	limitMove	The maximum daily change in price for the ASI.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`WilderAccumulativeSwingIndex` - The WilderAccumulativeSwingIndex for the given parameters.

Definition at line 1927 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `WilderAccumulativeSwingIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class WilderAccumulativeSwingIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.asi = WilderAccumulativeSwingIndex(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.asi.Update(bar)

        if self.asi.IsReady:
            # The current value of self.asi is represented by self.asi.Current.Value
            self.Plot("WilderAccumulativeSwingIndex", "asi", self.asi.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class WilderAccumulativeSwingIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.asi = WilderAccumulativeSwingIndex(20)
        self.RegisterIndicator(self.symbol, self.asi, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.asi.IsReady:
            # The current value of self.asi is represented by self.asi.Current.Value
            self.Plot("WilderAccumulativeSwingIndex", "asi", self.asi.Current.Value)
```

The following reference table describes the `WilderAccumulativeSwingIndex` constructor:

INDICATORS

WilderAccumulativeSwingIndex() 1/2

```
WilderAccumulativeSwingIndex QuantConnect.Indicators.WilderAccumulativeSwingIndex (
decimal limitMove
)
```

Initializes a new instance of the `WilderAccumulativeSwingInde` class using the specified name.

Show Details

Parameters		
`decimal`	limitMove	A decimal representing the limit move value for the period.

Return

`WilderAccumulativeSwingIndex` - The new `WilderAccumulativeSwingIndex` indicator object.

Definition at line 56 of file Indicators/WilderAccumulativeSwingIndex.cs.

INDICATORS

WilderAccumulativeSwingIndex() 2/2

```
WilderAccumulativeSwingIndex QuantConnect.Indicators.WilderAccumulativeSwingIndex (
string name,
decimal limitMove
)
```

Initializes a new instance of the `WilderAccumulativeSwingInde` class using the specified name.

Show Details

Parameters		
`string`	name	The name of this indicator.
`decimal`	limitMove	A decimal representing the limit move value for the period.

Return

`WilderAccumulativeSwingIndex` - The new `WilderAccumulativeSwingIndex` indicator object.

Definition at line 66 of file Indicators/WilderAccumulativeSwingIndex.cs.

Visualization

The following image shows plot values of selected properties of `WilderAccumulativeSwingIndex` using the plotly library.

Supported Indicators

Wilder Moving Average

Introduction

This indicator represents the moving average indicator defined by Welles Wilder in his book: New Concepts in Technical Trading Systems.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using WWMA Indicator

To create an automatic indicators for `WilderMovingAverage` , call the `WWMA` helper method from the `QCAlgorithm` class. The `WWMA` method creates a `WilderMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```py
class WilderMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.wwma = self.WWMA(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.wwma.IsReady:
            # The current value of self.wwma is represented by self.wwma.Current.Value
            self.Plot("WilderMovingAverage", "wwma", self.wwma.Current.Value)
```

The following reference table describes the `WWMA` method:

INDICATORS

WWMA() 1/1

```
WilderMovingAverage QuantConnect.Algorithm.QCAlgorithm.WWMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a WilderMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose WMA we want.
`Int32`	period	The period of the WMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`WilderMovingAverage` - The WilderMovingAverage for the given parameters.

Definition at line 1886 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `WilderMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with time/number pair, or an `IndicatorDataPoint` . The indicator will only be ready after you prime it with enough data.

PY

```
class WilderMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.wwma = WilderMovingAverage(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.wwma.Update(bar.EndTime, bar.Close)

        if self.wwma.IsReady:
            # The current value of self.wwma is represented by self.wwma.Current.Value
            self.Plot("WilderMovingAverage", "wwma", self.wwma.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class WilderMovingAverageAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.wwma = WilderMovingAverage(20)
        self.RegisterIndicator(self.symbol, self.wwma, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.wwma.IsReady:
            # The current value of self.wwma is represented by self.wwma.Current.Value
            self.Plot("WilderMovingAverage", "wwma", self.wwma.Current.Value)
```

The following reference table describes the `WilderMovingAverage` constructor:

INDICATORS

WilderMovingAverage() 1/2

```
WilderMovingAverage QuantConnect.Indicators.WilderMovingAverage (
string name,
int period
)
```

Initializes a new instance of the WilderMovingAverage class with the specified name and period.

Show Details

Parameters		
string	name	The name of this indicator.
int	period	The period of the Wilder Moving Average.

Return

`WilderMovingAverage` - The new `WilderMovingAverage` indicator object.

Definition at line 33 of file Indicators/WilderMovingAverage.cs.

INDICATORS

WilderMovingAverage() 2/2

```
WilderMovingAverage QuantConnect.Indicators.WilderMovingAverage (
int period
)
```

Initializes a new instance of the WilderMovingAverage class with the default name and period.

Show Details

Parameters		
`int`	period	The period of the Wilder Moving Average.

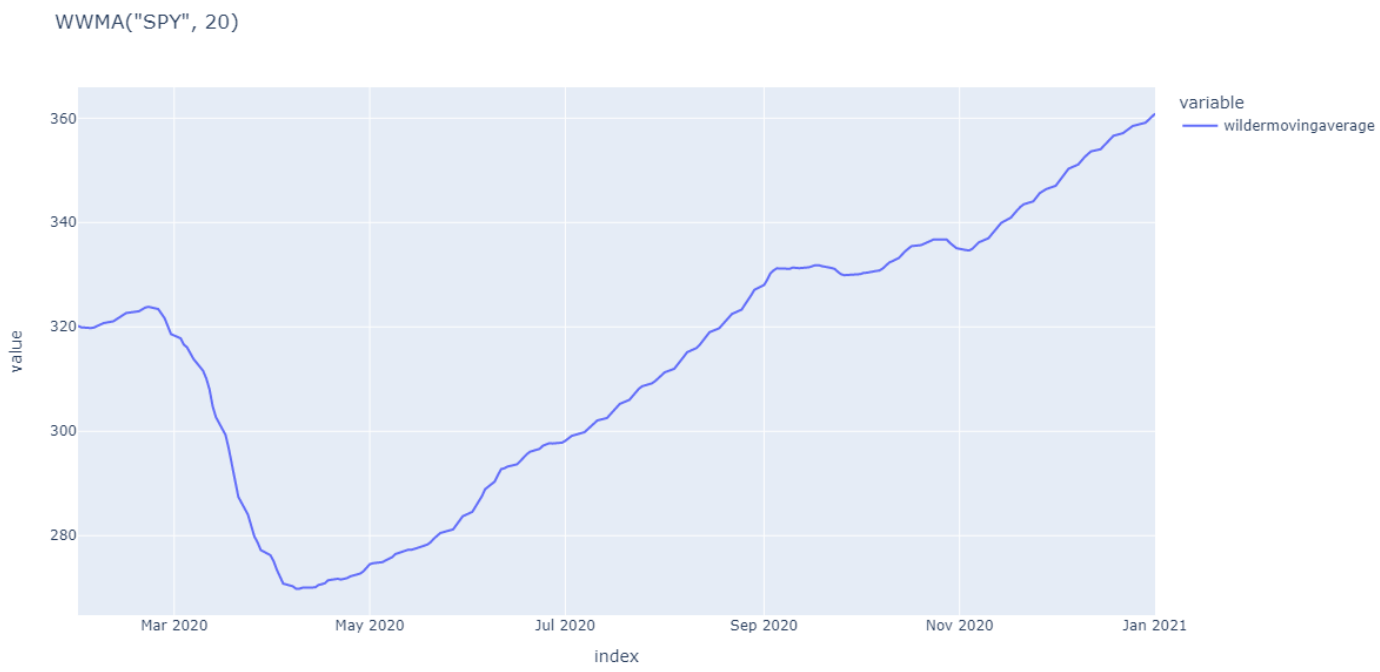
Return

`WilderMovingAverage` - The new `WilderMovingAverage` indicator object.

Definition at line 45 of file Indicators/WilderMovingAverage.cs.

Visualization

The following image shows plot values of selected properties of `WilderMovingAverage` using the plotly library.



Supported Indicators

Wilder Swing Index

Introduction

This indicator calculates the Swing Index (SI) as defined by Welles Wilder in his book 'New Concepts in Technical Trading Systems'. SI_t = 50 * (N / R) * (K / T) Where: N Equals: C_t - C_{t-1} + 0.5 * (C_t - O_t) + 0.25 * (C_{t-1} - O_{t-1}) *See* R Found by selecting the expression with the largest value and then using the corresponding formula. Expression => Formula $|H_t - C_{t-1}|$ => $|H_t - C_t|$ - 0.5 * $|L_t - C_{t-1}|$ + 0.25 * $|C_{t-1} - O_{t-1}|$ $|L_t - C_{t-1}|$ => $|L_t - C_t|$ - 0.5 * $|H_t - C_{t-1}|$ + 0.25 * $|C_{t-1} - O_{t-1}|$ $|H_t - L_t|$ => $|H_t - L_{t-1}|$ + 0.25 * $|C_{t-1} - O_{t-1}|$ *See* K Found by selecting the larger of the two expressions: $|H_t - C_{t-1}|$, $|L_t - C_{t-1}|$ *See* T The limit move, or the maximum change in price during the time period for the bar. Passed as limitMove via the constructor. *See*

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SI Indicator

To create an automatic indicators for `WilderSwingIndex` , call the `SI` helper method from the `QCAlgorithm` class. The `SI` method creates a `WilderSwingIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class WilderSwingIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.si = self.SI(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.si.IsReady:
            # The current value of self.si is represented by self.si.Current.Value
            self.Plot("WilderSwingIndex", "si", self.si.Current.Value)
```

The following reference table describes the `SI` method:

INDICATORS

SI() 1/1

```
WilderSwingIndex QuantConnect.Algorithm.QCAlgorithm.SI (
Symbol symbol,
Decimal limitMove,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a Wilder Swing Index (SI) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose SI we want.
`Decimal`	limitMove	The maximum daily change in price for the SI.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`WilderSwingIndex` - The WilderSwingIndex for the given parameters.

Definition at line 1906 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `WilderSwingIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` . The indicator will only be ready after you prime it with enough data.

```py
class WilderSwingIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.si = WilderSwingIndex(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.si.Update(bar)

        if self.si.IsReady:
            # The current value of self.si is represented by self.si.Current.Value
            self.Plot("WilderSwingIndex", "si", self.si.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

```py
class WilderSwingIndexAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.si = WilderSwingIndex(20)
        self.RegisterIndicator(self.symbol, self.si, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.si.IsReady:
            # The current value of self.si is represented by self.si.Current.Value
            self.Plot("WilderSwingIndex", "si", self.si.Current.Value)
```

The following reference table describes the `WilderSwingIndex` constructor:

INDICATORS

WilderSwingIndex() 1/2

```
WilderSwingIndex QuantConnect.Indicators.WilderSwingIndex (
string name,
decimal limitMove
)
```

Initializes a new instance of the `WilderSwingInde` class using the specified name.

Show Details

Parameters		
`string`	name	A string for the name of this indicator.
`decimal`	limitMove	A decimal representing the limit move value for the period.

Return

`WilderSwingIndex` - The new `WilderSwingIndex` indicator object.

Definition at line 116 of file Indicators/WilderSwingIndex.cs.

INDICATORS

WilderSwingIndex() 2/2

```
WilderSwingIndex QuantConnect.Indicators.WilderSwingIndex (
decimal limitMove
)
```

Initializes a new instance of the `WilderSwingInde` class using the default name.

Show Details

Parameters		
`decimal`	limitMove	A decimal representing the limit move value for the period.

Return

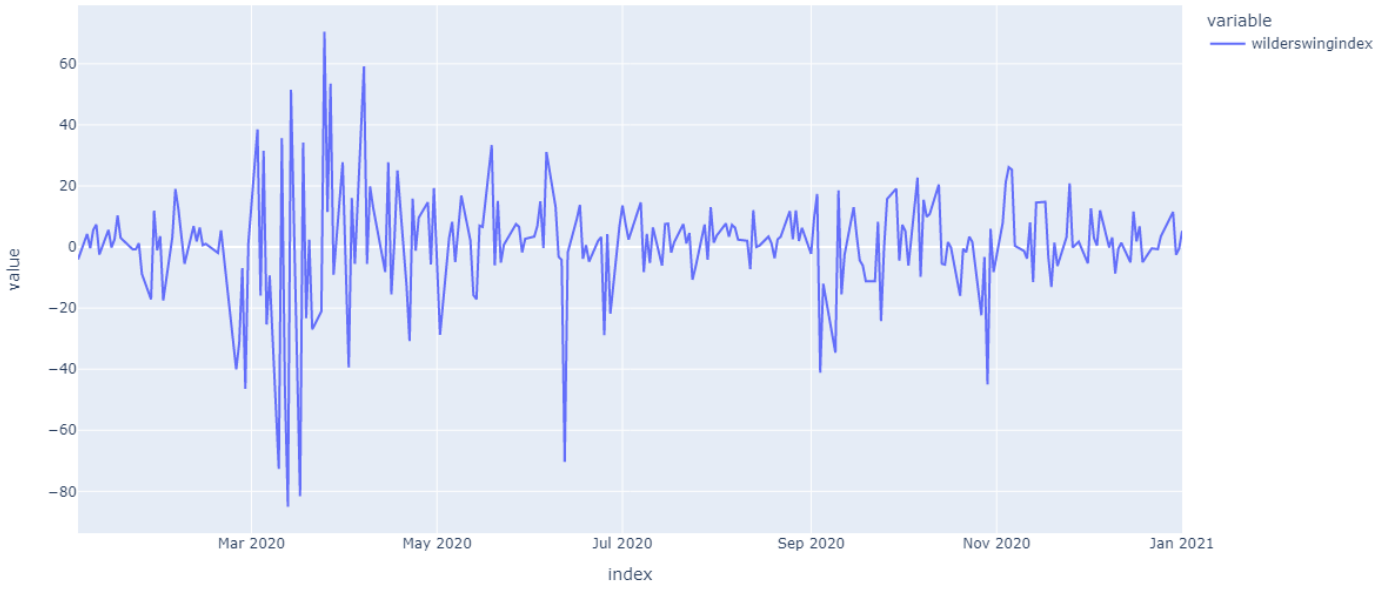
`WilderSwingIndex` - The new `WilderSwingIndex` indicator object.

Definition at line 126 of file Indicators/WilderSwingIndex.cs.

Visualization

The following image shows plot values of selected properties of `WilderSwingIndex` using the plotly library.

SI("SPY", 20)


variable
wilderswingindex
60
40
20
0
−20
−40
−60
−80
value
Mar 2020
May 2020
Jul 2020
Sep 2020
Nov 2020
Jan 2021
index

Supported Indicators

Williams Percent R

Introduction

Williams %R, or just %R, is the current closing price in relation to the high and low of the past N days (for a given N). The value of this indicator fluctuates between -100 and 0. The symbol is said to be oversold when the oscillator is below -80%, and overbought when the oscillator is above -20%.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using WILR Indicator

To create an automatic indicators for `WilliamsPercentR` , call the `WILR` helper method from the `QCAlgorithm` class. The `WILR` method creates a `WilliamsPercentR` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

PY

```
class WilliamsPercentRAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.wilr = self.WILR(self.symbol, 20)

    def OnData(self, slice: Slice) -> None:
        if self.wilr.IsReady:
            # The current value of self.wilr is represented by self.wilr.Current.Value
            self.Plot("WilliamsPercentR", "wilr", self.wilr.Current.Value)
            # Plot all attributes of self.wilr
            self.Plot("WilliamsPercentR", "maximum", self.wilr.Maximum.Current.Value)
            self.Plot("WilliamsPercentR", "minimum", self.wilr.Minimum.Current.Value)
```

The following reference table describes the `WILR` method:

INDICATORS

WILR() 1/1

```
WilliamsPercentR QuantConnect.Algorithm.QCAlgorithm.WILR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Williams %R indicator. This will compute the percentage change of the current closing price in relation to the high and low of the past N periods. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Williams %R we want.
`Int32`	period	The period over which to compute the Williams %R.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`WilliamsPercentR` - The Williams %R indicator for the requested symbol over the specified period.

Definition at line 1866 of file Algorithm/QCAlgorithm.Indicators.cs.

If you don't provide a resolution, it defaults to the security resolution. If you provide a resolution, it must be greater than or equal to the resolution of the security. For instance, if you subscribe to hourly data for a security, you should update its indicator with data that spans 1 hour or longer.

For more information about the selector argument, see Alternative Price Fields .

For more information about plotting indicators, see Plotting Indicators .

You can manually create a `WilliamsPercentR` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method with a `TradeBar` , or `QuoteBar` . The indicator will only be ready after you prime it with enough data.

PY

```
class WilliamsPercentRAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.wilr = WilliamsPercentR(20)

    def OnData(self, slice: Slice) -> None:
        bar = slice.Bars.get(self.symbol)
        if bar:
            self.wilr.Update(bar)

        if self.wilr.IsReady:
            # The current value of self.wilr is represented by self.wilr.Current.Value
            self.Plot("WilliamsPercentR", "wilr", self.wilr.Current.Value)
            # Plot all attributes of self.wilr
            self.Plot("WilliamsPercentR", "maximum", self.wilr.Maximum.Current.Value)
            self.Plot("WilliamsPercentR", "minimum", self.wilr.Minimum.Current.Value)
```

To register a manual indicator for automatic updates with the security data, call the `RegisterIndicator` method.

PY

```
class WilliamsPercentRAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.wilr = WilliamsPercentR(20)
        self.RegisterIndicator(self.symbol, self.wilr, Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        if self.wilr.IsReady:
            # The current value of self.wilr is represented by self.wilr.Current.Value
            self.Plot("WilliamsPercentR", "wilr", self.wilr.Current.Value)
            # Plot all attributes of self.wilr
            self.Plot("WilliamsPercentR", "maximum", self.wilr.Maximum.Current.Value)
            self.Plot("WilliamsPercentR", "minimum", self.wilr.Minimum.Current.Value)
```

The following reference table describes the `WilliamsPercentR` constructor:

INDICATORS

WilliamsPercentR() 1/2

```
WilliamsPercentR QuantConnect.Indicators.WilliamsPercentR (
int period
)
```

Creates a new Williams %R.

Show Details

Parameters		
int	period	The look-back period to determine the Williams %R.

Return

`WilliamsPercentR` - The new `WilliamsPercentR` indicator object.

Definition at line 52 of file Indicators/WilliamsPercentR.cs.

INDICATORS

WilliamsPercentR() 2/2

```
WilliamsPercentR QuantConnect.Indicators.WilliamsPercentR (
string name,
int period
)
```

Creates a new Williams %R.

Show Details

Parameters		
`string`	name	The name of this indicator.
`int`	period	The look-back period to determine the Williams %R.

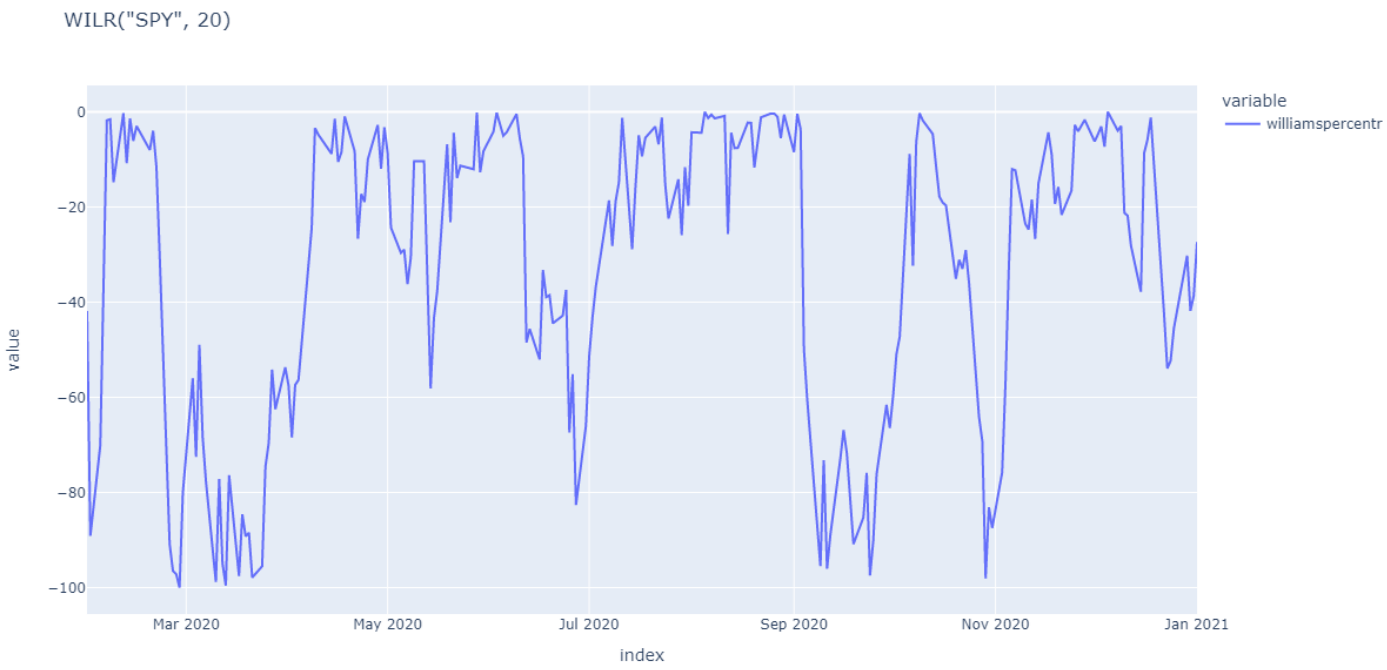
Return

`WilliamsPercentR` - The new `WilliamsPercentR` indicator object.

Definition at line 62 of file Indicators/WilliamsPercentR.cs.

Visualization

The following image shows plot values of selected properties of `WilliamsPercentR` using the plotly library.



Indicators

Key Concepts

Introduction

Indicators translate a stream of data points into a numerical value you can use to detect trading opportunities. LEAN provides more than 100 pre-built technical indicators and candlestick patterns you can use in your algorithms. To view a list of all the indicators, see the Supported Indicators . One simple indicator to learn is the Identity indicator, which just returns the value of the asset.

```
pep = self.Identity("PEP") # Pepsi ticker
```
PY

Design Philosophy

Technical indicators take in a stream of data points, `Bar` objects, or `TradeBar` objects and produce a continuous value. Candlestick patterns take in a stream of `Bar` or `TradeBar` objects and produce a value that's either 0, 1, or -1 to signify the presence of the pattern.

You can configure technical indicators and candlestick patterns to receive manual or automatic updates. With manual indicators, you update the indicator with whatever data you want and whenever you want. With automatic indicators, the algorithm automatically uses the security data to update the indicator value.

LEAN's indicators are implemented "on-line". This means as data pipes through, it updates the internal state of the indicator. Other platforms perform indicator math on an array of values in bulk which can be very inefficient when working on large volumes of data. You need to warm up your indicators and prime them with data to get them ready.

Indicator Types

You can classify an indicator as either a data point, bar, or TradeBar indicator. Their classification depends on the type of data they receive.

Data Point Indicators

Data point indicators use `IndicatorDataPoint` objects to compute their value. Some examples of data point indicators include the following:

- SimpleMovingAverage
- ExponentialMovingAverage
- Momentum

Bar Indicators

Bar indicators use `QuoteBar` or `TradeBar` objects to compute their value. Since Forex and CFD securities don't have `TradeBar` data, they use bar indicators. Candlestick patterns are examples of bar indicators. Some other examples of bar indicators include the following:

- WilliamsPercentR
- Stochastics
- AverageTrueRange

TradeBar Indicators

`TradeBar` indicators use `TradeBar` objects to compute their value. Some `TradeBar` indicators use the volume property of the `TradeBar` to compute their value. Some examples of `TradeBar` indicators include the following:

- Beta
- AdvanceDeclineVolumeRatio
- VolumeWeightedAveragePriceIndicator

Naming Convention

The class name to create a manual indicator is the indicator name spelled in pascal case. For example, to create a manual simple moving average indicator , use the `SimpleMovingAverage` class.

The method to create an automatic indicator is usually named after the acronym of the indicator name. For example, to create an automatic simple moving average indicator , use the `SMA` method.

To view the class name and shortcut method of each indicator, see the Supported Indicators .

Create Indicators

There are two ways to create indicators. You can create indicators with the indicator constructor or use a helper method in the `QCAlgorithm` class. If you use the helper method, the indicator automatically updates as your algorithm receives new data.

```
PY
# Create a manual indicator with the indicator constructor
self.manual_sma = SimpleMovingAverage(20, Resolution.Daily)

# Create an automatic indicator with the helper method
symbol = self.AddCrypto("BTCUSD").Symbol;
auto_sma = self.SMA(symbol, 20, Resolution.Daily)
```

Check Readiness

Indicators aren't always ready when you first create them. The length of time it takes to trust the indicator values depends on the indicator period. To know if an indicator is ready to use, use the `IsReady` property.

```
PY
if not self.indicator.IsReady:
    return
```

To get the number of samples the indicator has processed, use the `Samples` property.

```
PY
samples = self.indicator.Samples
```

Track Update Events

When an indicator receives a new data point and updates its state, it triggers an update event. To be notified of update events, attach an event handler to the indicator object. The event handler receives 2 arguments: the indicator object and the latest `IndicatorDataPoint` it produced.

```py
# After you create the indicator, attach an event handler
self.indicator.Updated += self.update_event_handler

# Define the event handler within the same class
def update_event_handler(self, indicator: object, indicator_data_point: IndicatorDataPoint) -> None:
    if indicator.IsReady:
        self.Plot("Indicator", "Value", indicator_data_point.Value)
```

Get Indicator Values

You can access current and historical indicator values.

Current Indicator Values

To access the indicator value, use the `.Current.Value` property. Some indicators have one output and some indicators have multiple outputs. The `SimpleMovingAverage` indicator only has one output, the average price over the last n periods, so the `.Current.Value` property returns this value. The `BollingerBand` indicator has multiple outputs because it has a simple moving average, an upper band, and a lower band. For indicators that have multiple outputs, refer to the Supported Indicators to see how to access the output values.

```py
sma = self.sma.Current.Value

current_price = self.bb.Current.Value
bb_upper_band = self.bb.UpperBand.Current.Value
bb_lower_band = self.bb.LowerBand.Current.Value
```

You can implicitly cast indicators to the decimal version of their `.Current.Value` property.

```py
if self.sma > self.bb.UpperBand:
    self.SetHoldings(self.symbol, -0.1)
```

Historical Indicator Values

To track historical indicator values, use a `RollingWindow` . Indicators emit an `Updated` event when they update. To create a `RollingWindow` of indicator points, attach an event handler function to the `Updated` member that adds the last value of the indicator to the `RollingWindow` . The value is an `IndicatorDataPoint` object that represents a piece of data at a specific time.

```py
def Initialize(self) -> None:
    # Creates an indicator and adds to a RollingWindow when it is updated
    self.sma_window = RollingWindow[IndicatorDataPoint](5)
    self.SMA("SPY", 5).Updated += (lambda sender, updated: self.sma_window.Add(updated))
```

The current (most recent) indicator value is at index 0, the previous value is at index 1, and so on until the length of the window.

```
current_sma = self.sma_window[0]
previous_sma = self.sma_window[1]
oldest_sma = self.sma_window[sma_window.Count - 1]
```

Reset Indicators

To reset indicators, call the `Reset` method.

```
self.indicator.Reset()
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your indicators when splits and dividends occur. If a split or dividend occurs, the data in your indicators becomes invalid because it doesn't account for the price adjustments that the split or dividend causes. To replace your indicator history with the newly-adjusted prices, call the `Reset` method and then warm up the indicator.

```
# In OnData
if data.Splits.ContainsKey(self.symbol) or data.Dividends.ContainsKey(self.symbol):
    # Reset the indicator
    self.sma.Reset()

    # Warm up the indicator
```

The process to warm up your indicator depends on if it's a manual or automatic indicator .

Common Design Patterns

The location in your algorithm where you create and manage indicators depends on the type of universe in your algorithm.

One Asset In a Static Universe

If you only have one security in your algorithm, you can create an automatic indicator in the `Initialize` method.

```
def Initialize(self):
  self.symbol = self.AddEquity("SPY").Symbol
    self.EnableAutomaticIndicatorWarmUp = True
    self.sma = self.SMA(self.symbol, 20, Resolution.Daily)
```

Multiple Assets or a Dynamic Universe

If your algorithm has multiple assets or a dynamic universe of assets, abstract your indicator management logic into a separate class.

```
class SymbolData:
    def __init__(self, algorithm, symbol):
        self.algorithm = algorithm
        self.symbol = symbol

        # Create an indicator
        self.sma = SimpleMovingAverage(20)

        # Create a consolidator to update the indicator
        self.consolidator = TradeBarConsolidator(1)
        self.consolidator.DataConsolidated += self.OnDataConsolidated

        # Register the consolidator to update the indicator
        algorithm.SubscriptionManager.AddConsolidator(symbol, self.consolidator)

        # Warm up the indicator
        algorithm.WarmUpIndicator(symbol, self.sma)

    def OnDataConsolidated(self, sender: object, consolidated_bar: TradeBar) -> None:
        self.sma.Update(consolidated_bar.EndTime, consolidated_bar.Close)

    # If you have a dynamic universe, remove consolidators for the securities removed from the universe
    def dispose(self) -> None:
        self.algorithm.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

Every time the universe adds or removes a security, create or dispose of the respective `SymbolData` object in the `OnSecuritiesChanged` event handler.

```
class MyAlgorithm(QCAlgorithm):
    symbol_data_by_symbol = {}

    def OnSecuritiesChanged(self, changes: SecurityChanges) -> None:
        for security in changes.AddedSecurities:
            self.symbol_data_by_symbol[security.Symbol] = SymbolData(self, security.Symbol)

        # If you have a dynamic universe, track removed securities
        for security in changes.RemovedSecurities:
            symbol_data = self.symbol_data_by_symbol.pop(security.Symbol, None)
            if symbol_data:
                symbol_data.dispose()
```

Differences From Third-Party Indicators

The values of our indicators can sometimes produce different results than the indicators on other platforms. These discrepancies can be a result of differences in the input data, timestamps, or implementation.

Price Differences

If you find differences in indicator values, compare the prices that are fed into the indicator on each platform. If the input data is slightly different, the indicator values will be different. To test if it's a difference in price data, feed in the data from the third-party platform into our indicators. We validate the indicators against third-party sources and when the values are the same, the indicator values are similar too.

Timestamp Differences

We timestamp our data to the time when the period ends. Many other platforms timestamp to the beginning of the candle.

Implementation Differences

Some indicators can have slightly different default arguments for their implementation. A common difference across platforms is to use a different `MovingAverageType` for the indicators. To view the default arguments for all our indicators, see Supported Indicators . To view the full implementation of our indicators, see the LEAN GitHub repository .

Warm Up Period Differences

Some platforms use all of the historical data to warm up their indicators while LEAN indicators have a fixed number of bars they need to warm up. As a result, indicators with long memory like the Exponential Moving Average can have slightly different values across platforms.

Examples

Demonstration Algorithms
IndicatorSuiteAlgorithm.py Python EmaCrossUniverseSelectionAlgorithm.py Python MACDTrendAlgorithm.py Python
RegressionChannelAlgorithm.py Python TalibIndicatorsAlgorithm.py Python

Indicators

Manual Indicators

Introduction

Manual indicators are indicators that don't automatically update from the underlying security data. Manual updates let you customize how you update the indicator and let you use the indicator in dynamic universes without causing your algorithm to slow down over time. It's not always necessary to create manual indicators though. If your algorithm only uses a static universe and you want to update the indicator with data from the security subscription, create an automatic indicator . Automatic indicators are easier to create and update.

Naming Convention

The class name to create a manual indicator is the indicator name spelled in pascal case. For example, to create a manual simple moving average indicator , use the `SimpleMovingAverage` class.

Create Manual Indicators

To create manual indicators, instantiate an indicator with its constructor. To view all of the available indicators and their constructors, see Supported Indicators .

```
PY
# Create an indicator
self.delay = Delay(20)

# Create a candlestick pattern
self.two_crows = TwoCrows()
```

You can track indicators by their name. To name a manual indicator, pass a string as the first argument to the constructor.

```
PY
# Name an indicator
self.delay = Delay("AAPL Past Price", 20)
self.Log(self.delay.Name)

# Name a candlestick pattern
self.two_crows = TwoCrows("Two Crows Pattern")
self.Log(self.two_crows.Name)
```

If you don't name an indicator, it's given a default name. For example, the default name for a `Delay` indicator is "Delay(period)".

Manual Updates

With manual updates, you control what data you use to update the indicator. For instance, you can use the 3:30 PM price in your daily moving average instead of the daily closing price or you can use the maximum temperature of the past 10 cloudy days.

To update an indicator, call the `Update` method. The `Update` method expects one of the following arguments:

- Time/decimal pair
- `IndicatorDataPoint`
- `QuoteBar`
- `TradeBar`
- Custom data bar

To view what data type you should use to update an indicator, see Supported Indicators .

You can update indicators at any point in your algorithm, but the most common places are during the `OnData` event handler or during a consolidation event handler.

```py
def Initialize(self) -> None:
    self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol

    self.rsi = RelativeStrengthIndex(10, MovingAverageType.Simple)
    self.ad = AccumulationDistribution()

    self.consolidator = TradeBarConsolidator(timedelta(days=3))
    # Update the AD indicator with the consolidated bar
    self.consolidator.DataConsolidated += (lambda _, bar : self.ad.Update(bar))
    self.SubscriptionManager.AddConsolidator(self.symbol, self.consolidator)

def OnData(self, slice: Slice) -> None:
    # Update the RSI indicator value with the new input close price every day
    if slice.Bars.ContainsKey(self.symbol):
        bar = slice.Bars[self.symbol]
        self.rsi.Update(bar.EndTime, bar.Close)
```

Automatic Updates

With automatic updates, your indicators automatically update with the security data on a schedule you set. To configure automatic updates, create a consolidator and then call the `RegisterIndicator` method. If your algorithm has a dynamic universe, save a reference to the consolidator so you can remove it when the universe removes the security. If you register an indicator for automatic updates, don't call the indicator's `Update` method or else the indicator will receive double updates.

```py
# Create a security subscription
self.symbol = self.AddEquity("SPY", Resolution.Minute).Symbol

# Create a manual indicator
self.indicator = RelativeStrengthIndex(10, MovingAverageType.Simple)

# Create a consolidator
self.consolidator = TradeBarConsolidator(1)

# Register the indicator to update with the consolidated data
self.RegisterIndicator(self.symbol, self.indicator, self.consolidator)
```

Data point indicators use only a single price data in their calculations. By default, those indicators use the closing price. For assets with `TradeBar` data, that price is the `TradeBar` close price. For assets with `QuoteBar` data, that price is the mid-price of the bid closing price and the ask closing price. To create an indicator with the other fields like the `Open` , `High` , `Low` , or `Close` , provide a `selector` argument to the `RegisterIndicator` method.

```
self.RegisterIndicator(self.symbol, self.indicator, self.consolidator, Field.High)
```

The `Field` class has the following `selector` properties:

Warm Up Indicators

Indicators use historical data to compute their value. Before you start trading with an indicator, warm it up. There are several ways to warm-up manual indicators.

Manual Indicator Warm-up

If you have access to the `QCAlgorithm` object, you can manually warmup indicators with a history request .

```
self.sma = SimpleMovingAverage(20)
history = algorithm.History(self.symbol, 20, Resolution.Daily)
if not history.empty:
    for time, row in history.loc[self.symbol].iterrows():
        self.sma.Update(time, row.close)
```

Warm-up Helper Method

If an indicator inherits the `IIndicatorWarmUpPeriodProvider` class, you can warm it up with the `WarmUpIndicator` method.

```
self.sma = SimpleMovingAverage(20)
algorithm.WarmUpIndicator(self.symbol, self.sma)
```

To warm up the indicator with a resolution that's different from the security resolution, pass a resolution or `timedelta` argument to the `WarmUpIndicator` method. The resolution you provide should be greater than or equal to the security resolution. For example, if the security has minute resolution data, you should warm up the indicator with data that spans at least one minute.

```
# Warm-up with daily bars
algorithm.WarmUpIndicator(self.symbol, self.sma, Resolution.Daily)

# Warm-up with 3-day bars
algorithm.WarmUpIndicator(self.symbol, self.sma, timedelta(days=3))
```

The `WarmUpIndicator` method uses the default `Value` of the historical data to warm up the indicator. In most cases, this is the closing price. To warm up the indicator with a different data field, pass a `Field` argument to the method.

```
algorithm.WarmUpIndicator(self.symbol, self.sma, Resolution.Daily, Field.High)
```

Algorithm Warm-up

If you create indicators at the beginning of your algorithm, you can set an algorithm warm-up period to warm up the

indicators. When you set an algorithm warm-up period, the engine pumps data in and automatically updates all the indicators from before the start date of the algorithm. To ensure that all the indicators are ready after the algorithm warm-up period, choose a lookback period that contains sufficient data.

```
# In Initialize
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
self.sma = SimpleMovingAverage(20)
self.SetWarmUp(20)

# In OnData
self.sma.Update(data[self.symbol]) # Delete this line if you registered the indicator for automatic
updates
if self.IsWarmingUp:
    return
```

PY

Timing Considerations

In some cases, you might create and warm up the indicator during its sampling period. For example, say the security resolution is minute, the indicator resolution is daily, and you create and warm-up the indicator at noon without using the `SetWarmUp` method. In this example, the history request that gathers data to warm up the indicator won't contain any data from the current day. Furthermore, if you set up a consolidator to update the indicator, the consolidator also won't aggregate any data from before noon. It doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Indicators

Automatic Indicators

Introduction

Automatic indicators are indicators that automatically update from the underlying security data. Automatic indicators do everything that manual indicators do, but with automatic updates registered for you. If you have a dynamic universe, you shouldn't use automatic indicators because you can't remove the consolidator that updates the indicator. Since you can't remove the consolidator, the consolidators build up over time and slow down your algorithm. If your algorithm has a dynamic universe, use manual indicators instead.

Naming Convention

The method to create an automatic indicator is usually named after the acronym of the indicator name. For example, to create an automatic simple moving average indicator , use the `SMA` method.

Create Automatic Indicators

To create automatic indicators, call the indicator helper method from the `QCAlgorithm` class. The indicator helper methods create an indicator object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. To view the helper method for each indicator, see Supported Indicators . In most cases, you should call the helper method in the `Initialize` method.

The indicator resolution must be greater than or equal to the resolution of the security subscription. For instance, if your security subscription is for minute resolution data, the indicator resolution should be minute, hour, or daily resolution.

```
PY
# In Initialize()
symbol = self.AddEquity("SPY").Symbol

#  - Create an indicator
self.sma = self.SMA(symbol, 20, Resolution.Daily)

#  - Create a candlestick pattern
patterns = CandlestickPatterns(self)
self.two_crows = patterns.TwoCrows(self.symbol)
```

When you create an indicator with a helper method, the indicator is given a default name. You can track indicators by their name. The default name for the `SimpleMovingAverage` indicator is "SMA(period , ticker _ resolution)". For example, in the preceding code snippet, the `SimpleMovingAverage` indicator is named "SMA(20,SPY_day)". To get the name of an indicator, inspect its `Name` property.

```
PY
self.Log(self.sma.Name)
self.Log(self.two_crows.Name)
```

Alternative Price Fields

Data point indicators use only a single price data in their calculations. By default, those indicators use the closing price. For assets with `TradeBar` data, that price is the `TradeBar` close price. For assets with `QuoteBar` data, that price is the mid-price of the bid closing price and the ask closing price. To create an indicator with the other fields like the `Open` , `High` , `Low` , or `Close` , provide a `selector` argument to the indicator helper method.

```
# Define a 10-period daily RSI indicator with shortcut helper method
# Select the Open price to update the indicator
self.rsi = self.RSI("SPY", 10,  MovingAverageType.Simple, Resolution.Daily, Field.Open)
```

The `Field` class has the following `selector` properties:

Warm Up Indicators

Indicators compute their value based on historical data. Before you start trading with an indicator, warm it up. There are several ways to warm-up automatic indicators.

Algorithm Warm-up

You can set an algorithm warm-up period to warm up the indicators. When you set an algorithm warm-up period, the engine pumps data in and automatically updates all the indicators from before the start date. To ensure that all the indicators are ready after the algorithm warm-up period, choose a look-back period that contains sufficient data.

```
# In Initialize
symbol = self.AddEquity("SPY").Symbol
self.sma = self.SMA(symbol, 20, Resolution.Daily)
self.SetWarmUp(20, Resolution.Daily)

# In OnData
if self.IsWarmingUp:
    return
```

Manual Indicator Warm-up

You can manually warm up indicators with a history request .

```
# In Initialize
symbol = self.AddEquity("SPY").Symbol
self.sma = self.SMA(symbol, 20, Resolution.Daily)

history = self.History(symbol, 20, Resolution.Daily)
if not history.empty:
    for time, row in history.loc[symbol].iterrows():
        self.sma.Update(time, row.close)
```

Automatic Indicator Warm-up

You can set the `EnableAutomaticIndicatorWarmUp` property to true before you create indicators to automatically warm them up.

```
# In Initialize
symbol = self.AddEquity("SPY").Symbol
self.EnableAutomaticIndicatorWarmUp = True
self.sma = self.SMA(symbol, 20, Resolution.Daily)
```

Warm-up Helper Method

If an indicator inherits the `IIndicatorWarmUpPeriodProvider` class, you can warm it up with the `WarmUpIndicator` method.

```
self.sma = self.SMA(self.symbol, 20)
self.WarmUpIndicator(self.symbol, self.sma)
```

To warm up the indicator with a resolution that's different from the security resolution, pass a resolution or `timedelta` argument to the `WarmUpIndicator` method. The resolution you provide should be greater than or equal to the security resolution. For example, if the security has minute resolution data, you should warm up the indicator with data that spans at least one minute. If the indicator resolution is different from the security resolution, provide the indicator resolution as the argument to properly warm up the indicator.

```
# Warm up with daily bars
self.WarmUpIndicator(self.symbol, self.sma, Resolution.Daily)

# Warm up with 3-day bars
self.WarmUpIndicator(self.symbol, self.sma, timedelta(days=3))
```

The `WarmUpIndicator` method uses the default `Value` of the historical data to warm up the indicator. In most cases, this is the closing price. To warm up the indicator with a different data field, pass a `Field` argument to the method.

```
self.WarmUpIndicator(self.symbol, self.sma, Resolution.Daily, Field.High)
```

Timing Considerations

In some cases, you might create and warm up the indicator during its sampling period. For example, say the security resolution is minute, the indicator resolution is daily, and you create and warm-up the indicator at noon without using the `SetWarmUp` method. In this example, the history request that gathers data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. It doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Common Mistakes

Avoid the following common mistakes when you use automatic indicators.

Creating Automatic Indicators in a Dynamic Universe

You can't currently remove the consolidator that LEAN creates to update automatic indicators. If you add consolidators to a dynamic universe, the consolidators build up over time and slow down your algorithm. To avoid issues, if you algorithm has a dynamic universe, use manual indicators .

Using Indicator Values Before the Indicator Is Warmed Up

Indicators can provide inaccurate output values before they are warmed up. To avoid issues, always check the `IsReady` flag before you use indicator output values.

```
PY
if self.indicator.IsReady:
    value = self.indicator.Current.Value
```

Manually Registering an Automatic Indicator for Updates

If you create an automatic indicator and then register it for automatics updates or call the `Update` method, the indicator receives multiple input values during each update cycle. To avoid issues, create automatic indicators but don't register them for automatic updates or call the `Update` method.

```
PY
# Create automatic indicators
self.indicator = self.SMA(self.symbol, 5)

# Don't do this:
self.RegisterIndicator(self.symbol, self.indicator, Resolution.Daily)
```

Indicators

Plotting Indicators

Introduction

LEAN provides helper methods to make it simple to create indicator plots.

Plot Update Events

To plot all of the values of some indicators, in the `Initialize` method, call the `PlotIndicator` method. The method plots each indicator value as the indicator updates. The method accepts up to four indicators.

```py
symbol = self.AddEquity("SPY")
sma_short = self.SMA(symbol, 10)
sma_long = self.SMA(symbol, 20)
self.PlotIndicator("<chartName>", sma_short, sma_long)
```

Plot Current Values

To plot the current value of indicators, call the `Plot` method. The method accepts up to four indicators.

```py
# In Initialize
symbol = self.AddEquity("SPY")
sma_short = self.SMA(symbol, 10)
sma_long = self.SMA(symbol, 20)

# In OnData
self.Plot("<chartName>", sma_short, sma_long)
```

View Charts

The following table describes where you can access your charts, depending on how to deploy your algorithms:

Location	Algorithm Lab Algorithms	CLI Cloud Algorithms	CLI Local Algorithms
Backtest results page	✓	✓	
Live results page	✓	✓	
/backtests/read endpoint	✓	✓	
/live/read endpoint	✓	✓	
ReadBacktest method	✓	✓	
ReadLiveAlgorithm method	✓	✓	
Local **JSON** file in your **<projectName> / backtests / <timestamp>** or **<projectName> / live / <timestamp>** directory		✓	✓

Chart Limits

Not all indicators share the same base type(T), so you may not be able to plot them together since some indicators require points while others require `TradeBars` .

```py
# Plot indicators that extend the "Indicator" type
self.PlotIndicators("All Indicator Values", sma, rsi)
self.Plot("Current Indicator Values", sma, rsi);

# Plot indicators that extend the "TradeBarIndicator" type
self.PlotIndicators("All Indicator Values", atr, aroon)
self.Plot("Current Indicator Values", atr, aroon);
```

If your indicator plots are complex, call the `Plot` method with one indicator and plot its `.Current.Value` . For more information about plotting, see Charting .

Indicators

Combining Indicators

Introduction

Indicator extensions let you chain indications together like Lego blocks to create unique combinations. When you chain indicators together, the `.Current.Value` output of one indicator is the input of the following indicator. To chain indicators together with values other than the `.Current.Value` , create a custom indicator .

Addition

The `Plus` extension sums the `.Current.Value` of two indicators or sums the `.Current.Value` of an indicator and a fixed value.

```
# Sum the output of two indicators
rsi_short = self.RSI("SPY", 14)
rsi_long = self.RSI("SPY", 21)
rsi_plus_rsi = IndicatorExtensions.Plus(rsi_short, rsi_long)

# Sum the output of an indicator and a fixed value
rsi_plus_value = IndicatorExtensions.Plus(rsi_short, 10)
```

If you pass an indicator to the `Plus` extension, you can name the composite indicator.

```
named_indicator = IndicatorExtensions.Plus(rsi_short, rsi_long, "RSI Sum")
```

Subtraction

The `Minus` extension subtracts the `.Current.Value` of two indicators or subtracts a fixed value from the `.Current.Value` of an indicator.

```
# Subtract the output of two indicators
sma_short = self.SMA("SPY", 14)
sma_long = self.SMA("SPY", 21)
sma_difference = IndicatorExtensions.Minus(sma_short, sma_long)

# Subtract a fixed value from the output of an indicator
sma_minus_value = IndicatorExtensions.Minus(sma_short, 10)
```

If you pass an indicator to the `Minus` extension, you can name the composite indicator.

```
named_indicator = IndicatorExtensions.Minus(sma_short, sma_long, "SMA Difference")
```

Multiplication

The `Times` extension multiplies the `.Current.Value` of two indicators or multiplies a fixed value and the `.Current.Value` of an indicator.

```py
# Multiply the output of two indicators
ema_short = self.EMA("SPY", 14)
ema_long = self.EMA("SPY", 21)
ema_product = IndicatorExtensions.Times(ema_short, ema_long)

# Multiply the output of an indicator and a fixed value
ema_times_value = IndicatorExtensions.Times(ema_short, 1.5)
```

If you pass an indicator to the `Times` extension, you can name the composite indicator.

```py
named_indicator = IndicatorExtensions.Times(ema_short, ema_long, "EMA Product")
```

Division

The `Over` extension divides the `.Current.Value` of an indicator by the `.Current.Value` of another indicator or a fixed value.

```py
# Divide the output of two indicators
rsi_short = self.RSI("SPY", 14)
rsi_long = self.RSI("SPY", 21)
rsi_division = rsi_short.Over(rsi_long)

# Divide the output of an indicator by a fixed value
rsi_half = IndicatorExtensions.Over(rsi_short, 2)
```

If you pass an indicator to the `Over` extension, you can name the composite indicator.

```py
named_indicator = IndicatorExtensions.Over(rsi_short, rsi_long, "RSI Division")
```

Weighted Average

The `WeightedBy` extension calculates the average `.Current.Value` of an indicator over a lookback period, weighted by another indicator over the same lookback period. The value of the calculation is

$$\frac{\textbf{x} \cdot \textbf{y}}{ \sum\limits_{i=1}^{n} y_{i} }$$

where $\textbf{x}$ is a vector that contains the historical values of the first indicator, $\textbf{y}$ is a vector that contains the historical values of the second indicator, and n is the lookback period.

```py
sma_short = self.SMA("SPY", 14)
sma_long = self.SMA("SPY", 21)
weighted_sma = IndicatorExtensions.WeightedBy(sma_short, sma_long, 3)
```

Custom Chains

The `Of` extension feeds an indicator's `.Current.Value` into the input of another indicator. The first argument of the

`IndicatorExtensions.Of` method must be a manual indicator with no automatic updates. If you pass an indicator that has automatic updates as the argument, that first indicator is updated twice. The first update is from the security data and the second update is from the `IndicatorExtensions` class.

```
rsi = self.RSI("SPY", 14)
rsi_sma = IndicatorExtensions.Of(SimpleMovingAverage(10), rsi) # 10-period SMA of the 14-period RSI
```

If you pass a manual indicator as the second argument, to update the indicator chain, update the second indicator. If you call the `Update` method of the entire indicator chain, it won't update the chain properly.

Simple Moving Average

The `SMA` extension calculates the simple moving average of an indicator's `.Current.Value` .

```
rsi = self.RSI("SPY", 14) # Create a RSI indicator
rsi_sma = IndicatorExtensions.SMA(rsi, 3) # Create an indicator to calculate the 3-period SMA of the RSI
indicator
```

Exponential Moving Average

The `EMA` extension calculates the exponential moving average of an indicator's `.Current.Value` .

```
rsi = self.RSI("SPY", 14) # Create a RSI indicator
rsi_ema = IndicatorExtensions.EMA(rsi, 3) # Create an indicator to calculate the 3-period EMA of the RSI
indicator
```

The `EMA` extension can also accept a smoothing parameter that sets the percentage of data from the previous value that's carried into the next value.

```
rsi_ema = IndicatorExtensions.EMA(rsi, 3, 0.1) # 10% smoothing factor
```

Maximum

The `MAX` extension calculates an indicator's maximum `.Current.Value` over a lookback window.

```
ema = self.EMA("SPY", 14) # Create an EMA indicator
ema_max = IndicatorExtensions.MAX(ema, 10) # Create an indicator to calculate the maximum EMA over the
last 10 periods
```

Minimum

The `MIN` extension calculates an indicator's minimum `.Current.Value` over a lookback window.

```
ema = self.EMA("SPY", 14) # Create an EMA indicator
ema_min = IndicatorExtensions.MIN(ema, 10) # Create an indicator to calculate the minimum EMA over the
last 10 periods
```

Examples

Demonstration Algorithm
DisplacedMovingAverageRibbon.py Python

Indicators

Custom Indicators

Introduction

LEAN supports over 100 pre-built indicators, but a custom indicator is an indicator you define. It receives input, performs some calculation, and sets its output value. Custom indicators are helpful when you want to achieve any of the following results:

- Use an indicator that LEAN doesn't currently implement
- Combine built-in indicators beyond the logic of Indicator Extensions
- Create your own unique indicator for trading

Define Indicators

Custom indicators must implement the `PythonIndicator` class. The indicator must have an `Update` method and `Name` , `Time` , and `Value` attributes. The `Update` method must accept an `IndicatorDataPoint` , `QuoteBar` , or `TradeBar` and return a boolean that represents if the indicator is ready. The `Time` attribute represents the last time you updated the indicator and the `Value` attribute represents the current indicator value. The following definition provides an example of a custom simple moving average indicator.

```
PY
class CustomSimpleMovingAverage(PythonIndicator):
    def __init__(self, name, period):
        self.Name = name
        self.WarmUpPeriod = period
        self.Time = datetime.min
        self.Value = 0
        self.queue = deque(maxlen=period)

    def Update(self, input: BaseData) -> bool:
        self.queue.appendleft(input.Value)
        count = len(self.queue)
        self.Time = input.Time
        self.Value = sum(self.queue) / count
        return count == self.queue.maxlen
```

Create Indicators

You must define a custom indicator before you can create an instance of it.

To create a custom indicator, call the indicator constructor.

```
PY
self.custom_sma = CustomSimpleMovingAverage("My SMA", 10)
```

Updates

The process to update custom indicators is the same process you use to update manual indicators. For more information about updating manual indicators, see Manual Updates or Automatic Updates .

Warm Up Indicators

The process to warm up custom indicators is the same process you use to warm up manual indicators .

Examples

Demonstration Algorithms
CustomIndicatorAlgorithm.py Python

Indicators

Indicator Universes

Introduction

An indicator universe uses technical indicators to determine the constituents of the universe. Imagine a universe that only contains assets above their 10-day simple moving average. You can incorporate indicators into any of the types of universes in the Universes chapter . To create an indicator universe, define a helper class that contains the indicators and then define a universe that updates the indicators and selects assets.

Define SymbolData Objects

To make it easy to create and update indicators for each security in the universe, move the indicator logic into a class. In the universe definition, you can create an instance of this class for each security in the universe.

```
class SymbolData(object):
    def __init__(self, symbol):
        self.symbol = symbol
        self.tolerance = 1.01
        self.fast = ExponentialMovingAverage(100)
        self.slow = ExponentialMovingAverage(300)
        self.is_uptrend = False
        self.scale = 0

    def update(self, time, value):
        if self.fast.Update(time, value) and self.slow.Update(time, value):
            fast = self.fast.Current.Value
            slow = self.slow.Current.Value
            self.is_uptrend = fast > slow * self.tolerance

        if self.is_uptrend:
            self.scale = (fast - slow) / ((fast + slow) / 2.0)
```

PY

Define the Universe

You need to define SymbolData objects before you define the universe that selects securities.

When your universe function receives an object that contains all the possible securities, create a `SymbolData` object for each new security and update the remaining `SymbolData` objects with their daily price or some other data point. For example, the following universe definition selects US Equities that have the greatest difference between two moving averages.

```py
class EmaCrossUniverseSelectionAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        '''Initialise the data and resolution required, as well as the cash and start-end dates for your
algorithm. All algorithms must initialized.'''

        self.SetStartDate(2010,1,1)  #Set Start Date
        self.SetEndDate(2015,1,1)    #Set End Date
        self.SetCash(100000)           #Set Strategy Cash

        self.UniverseSettings.Resolution = Resolution.Daily
        self.UniverseSettings.Leverage = 2

        self.coarse_count = 10
        self.averages = { }

        # this add universe method accepts two parameters:
        # - coarse selection function: accepts an IEnumerable<CoarseFundamental> and returns an
IEnumerable<Symbol>
        self.AddUniverse(self.CoarseSelectionFunction)


    # sort the data by daily dollar volume and take the top 'NumberOfSymbols'
    def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:

        # We are going to use a dictionary to refer the object that will keep the moving averages
        for cf in coarse:
            if cf.Symbol not in self.averages:
                self.averages[cf.Symbol] = SymbolData(cf.Symbol)

            # Updates the SymbolData object with current EOD price
            avg = self.averages[cf.Symbol]
            avg.update(cf.EndTime, cf.AdjustedPrice)

        # Filter the values of the dict: we only want up-trending securities
        values = list(filter(lambda x: x.is_uptrend, self.averages.values()))

        # Sorts the values of the dict: we want those with greater difference between the moving averages
        values.sort(key=lambda x: x.scale, reverse=True)

        for x in values[:self.coarse_count]:
            self.Log('symbol: ' + str(x.symbol.Value) + '  scale: ' + str(x.scale))

        # we need to return only the symbol objects
        return [ x.symbol for x in values[:self.coarse_count] ]
```

Examples

Demonstration Algorithms
EmaCrossUniverseSelectionAlgorithm.py Python

Indicators

Rolling Window

Introduction

A `RollingWindow` is an array of a fixed-size that holds trailing data. It's more efficient to use `RollingWindow` objects to hold periods of data than to make multiple historical data requests . With a `RollingWindow` , you just update the latest data point while a `History` call fetches all of the data over the period you request. `RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window.

Supported Types

`RollingWindow` objects can store any native or C# types.

```
PY
self.close_window = RollingWindow[float](4)
self.trade_bar_window = RollingWindow[TradeBar](2)
self.quote_bar_window = RollingWindow[QuoteBar](2)
```

To be notified when `RollingWindow` objects support additional types, subscribe to GitHub Issue #6199 .

Add Data

To add data to a `RollingWindow` , call the `Add` method.

```
PY
self.close_window.Add(data["SPY"].Close)
self.trade_bar_window.Add(data["SPY"])
self.quote_bar_window.Add(data["EURUSD"])
```

Warm Up

To warm up a `RollingWindow` , make a history request and then iterate through the result to add the data to the `RollingWindow` .

```
PY
spy = self.AddEquity("SPY", Resolution.Daily).Symbol
history_trade_bar = self.History[TradeBar](spy, 10, Resolution.Daily)
history_quote_bar = self.History[QuoteBar](spy, 10, Resolution.Minute)

# Warm up the close price and trade bar rolling windows with the previous 10-day trade bar data
close_price_window = RollingWindow[float](10)
trade_bar_window = RollingWindow[TradeBar](10)
for trade_bar in history_trade_bar:
    close_price_window.Add(trade_bar.Close)
    trade_bar_window.Add(trade_bar)

# Warm up the quote bar rolling window with the previous 10-minute quote bar data
quote_bar_window = RollingWindow[QuoteBar](10)
for quote_bar in history_quote_bar:
    quote_bar_window.Add(quote_bar)
```

Check Readiness

To check if a `RollingWindow` is full, use its `IsReady` flag.

```
PY
if not self.close_window.IsReady:
    return
```

Access Data

`RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window.

```
PY
current_close = self.close_window[0]
previous_close = self.close_window[1]
oldest_close = self.close_window[self.close_window.Count-1]
```

To get the item that was most recently removed from the `RollingWindow` , use the `MostRecentlyRemoved` property.

```
PY
removed_close = self.close_window.MostRecentlyRemoved
```

Combine with Indicators

To track historical indicator values, use a `RollingWindow` . Indicators emit an `Updated` event when they update. To create a `RollingWindow` of indicator points, attach an event handler function to the `Updated` member that adds the last value of the indicator to the `RollingWindow` . The value is an `IndicatorDataPoint` object that represents a piece of data at a specific time.

```
PY
def Initialize(self) -> None:
    # Creates an indicator and adds to a RollingWindow when it is updated
    self.sma_window = RollingWindow[IndicatorDataPoint](5)
    self.SMA("SPY", 5).Updated += (lambda sender, updated: self.sma_window.Add(updated))
```

To view how to access individual members in an indicator, see Get Indicator Values .

```
PY
current_sma = self.sma_window[0]
previous_sma = self.sma_window[1]
oldest_sma = self.sma_window[sma_window.Count - 1]
```

Combine with Consolidators

To store a history of consolidated bars , in the consolidation handler, add the consolidated bar to the `RollingWindow` .

```
PY
self.consolidator.DataConsolidated += lambda sender, consolidated_bar:
self.trade_bar_window.Add(consolidated_bar)
```

Cast to Other Types

You can cast a `RollingWindow` to a list or a DataFrame. If you cast it to a list, reverse the list so the most recent element is at the last index of the list. This is the order the elements would be in if you added the elements to the list with the `Add` method. To cast a `RollingWindow` to a DataFrame, the `RollingWindow` must contain `Slice` , `Tick` , `QuoteBar` , or `TradeBar` objects. If the `RollingWindow` contains ticks, the ticks must have unique timestamps.

```
PY

closes = list(self.close_window)[::-1]

tick_df = self.PandasConverter.GetDataFrame[Tick](list(self.tick_window)[::-1])
trade_bar_df = self.PandasConverter.GetDataFrame[TradeBar](list(self.trade_bar_window)[::-1])
quote_bar_df = self.PandasConverter.GetDataFrame[QuoteBar](list(self.quote_bar_window)[::-1])
```

Delete Data

To remove all of the elements from a `RollingWindow` , call the `Reset` method.

```
PY

self.close_window.Reset()
```

Examples

Demonstration Algorithm
RollingWindowAlgorithm.py Python CustomVolatilityModelAlgorithm.py Python
MultipleSymbolConsolidationAlgorithm.py Python

Object Store

Introduction

The Object Store is a file system that you can use in your algorithms to save, read, and delete data. The Object Store is organization-specific, so you can save or read data from the same Object Store in all of your organization's projects. The Object Store works like a key-value storage system where you can store regular strings, JSON encoded strings, XML encoded strings, and bytes. You can access the data you store in the Object Store from backtests, the Research Environment, and live algorithms.

Get All Stored Data

To get all of the keys and values in the Object Store, iterate through the `ObjectStore` object.

```
PY
for kvp in self.ObjectStore:
    key = kvp.Key
    value = kvp.Value
```

To iterate through just the keys in the Object Store, iterate through the `Keys` property.

```
PY
for key in self.ObjectStore.Keys:
    continue
```

Create Sample Data

You need some data to store data in the Object Store.

Follow these steps to create some sample data:

1. Create a `string` .

   ```
   PY
   string_sample = "My string"
   ```

2. Create a `Bytes` object.

   ```
   PY
   bytes_sample = str.encode("My String")
   ```

Save Data

The Object Store saves objects under a key-value system. If you save objects in backtests, you can access them from the Research Environment. To avoid slowing down your backtests, save data once in the `OnEndOfAlgorithm` event handler. In live trading, you can save data more frequently like at the end of a `Train` method or after universe selection.

If you run algorithms in QuantConnect Cloud, you need storage create permissions to save data in the Object Store.

If you don't have data to store, create some sample data .

You can save `Bytes` and `string` objects in the Object Store. To store data, you need to provide a key. If you provide a key that is already in the Object Store, it will overwrite the data at that location. To avoid overwriting objects from other projects in your organization, prefix the key with your project ID. You can find the project ID in the URL of your browser when you open a project. For example, the ID of the project at **quantconnect.com/project/12345** is 12345.

Bytes

To save a `Bytes` object, call the `SaveBytes` method.

```
save_successful = self.ObjectStore.SaveBytes(f"{self.ProjectId}/bytes_key", bytes_sample)
```

Strings

To save a `string` object, call the `Save` or `SaveString` method.

```
save_successful = self.ObjectStore.Save(f"{self.ProjectId}/string_key", string_sample)
```

Read Data

Read data from the Object Store to import algorithm variables between deployments, import data from the Research Environment, or load trained machine learning models. To read data from the Object Store, you need to provide the key you used to store the object.

You can load `Bytes` and `string` objects from the Object Store.

Before you read data from the Object Store, check if the key exists.

```
if self.ObjectStore.ContainsKey(key):
    # Read data
```

Bytes

To read a `Bytes` object, call the `ReadBytes` method.

```
byte_data = self.ObjectStore.ReadBytes(f"{self.ProjectId}/bytes_key")
```

Strings

To read a `string` object, call the `Read` or `ReadString` method.

```
string_data = self.ObjectStore.Read(f"{self.ProjectId}/string_key")
```

Delete Data

Delete objects in the Object Store to remove objects that you no longer need. If you run algorithms in QuantConnect Cloud, you need storage delete permissions to delete data from the Object Store.

To delete objects from the Object Store, call the `Delete` method. Before you delete data, check if the key exists. If you try to delete an object with a key that doesn't exist in the Object Store, the method raises an exception.

```
PY
if self.ObjectStore.ContainsKey(key):
    self.ObjectStore.Delete(key)
```

To delete all of the content in the Object Store, iterate through all the stored data.

```
PY
for kvp in self.ObjectStore:
    self.ObjectStore.Delete(kvp.Key)
```

Cache Data

When you write to or read from the Object Store, the algorithm caches the data. The cache speeds up the algorithm execution because if you try to read the Object Store data again with the same key, it returns the cached data instead of downloading the data again. The cache speeds up execution, but it can cause problems if you are trying to share data between two nodes under the same Object Store key. For example, consider the following scenario:

1. You open project A and save data under the key `123` .
2. You open project B and save new data under the same key `123` .
3. In project A, you read the Object Store data under the key `123` , expecting the data from project B, but you get the original data you saved in step #1 instead.

   You get the data from step 1 instead of step 2 because the cache contains the data from step 1.

To clear the cache, call the `Clear` method.

```
PY
self.ObjectStore.Clear()
```

Get File Path

To get the file path for a specific key in the Object Store, call the `GetFilePath` method. If the key you pass to the method doesn't already exist in the Object Store, it's added to the Object Store.

```
PY
file_path = self.ObjectStore.GetFilePath(key)
```

Storage Quotas

If you run algorithms locally, you can store as much data as your hardware will allow. If you run algorithms in QuantConnect Cloud, you must stay within your storage quota . If you need more storage space, edit your storage plan

.

Example for Plotting

You can use the `ObjectStore` to plot data from your backtests and live algorithm in the Research Environment. In the following example, you will learn how to plot the Simple Moving Average indicator generated in a backtest.

1. Create a algorithm, add a data subscription and a Simple Moving Average indicator.

```
class ObjectStoreChartingAlgorithm(QCAlgorithm):
    def Initialize(self):
        self.AddEquity("SPY")

        self.content = ''
        self.sma = self.SMA("SPY", 22)
```

   The algorithm will save `self.content` to the `ObjectStore` .

2. Save indicator data as `string` in `self.content` .

```
def OnData(self, data: Slice):
    self.Plot('SMA', 'Value', self.sma.Current.Value)
    self.content += f'{self.sma.Current.EndTime},{self.sma.Current.Value}\n'
```

3. To store the collected data, call the `Save` method with a key.

```
def OnEndOfAlgorithm(self):
    self.ObjectStore.Save('sma_values_python', self.content)
```

4. Open the Research Environment, and create a `QuantBook` .

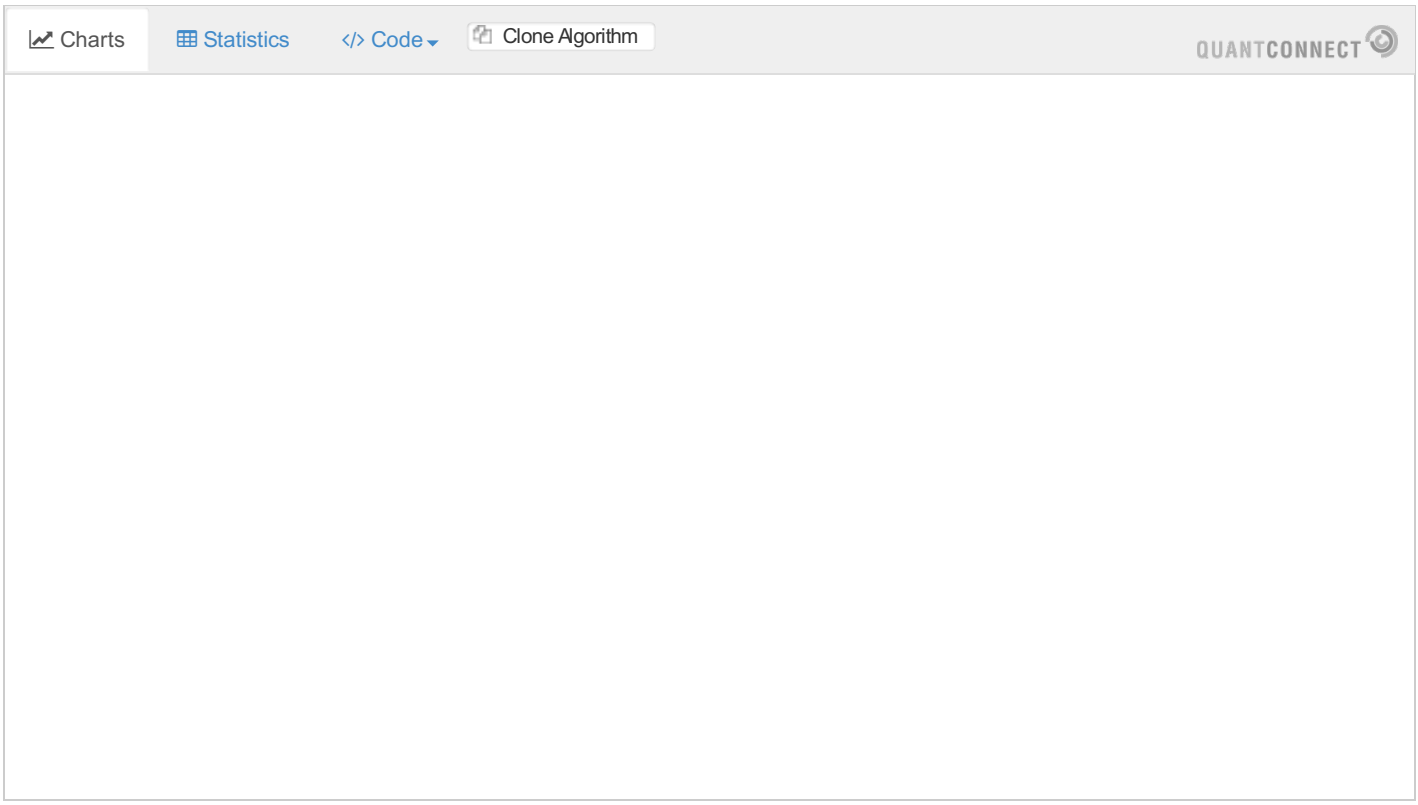
```
qb = QuantBook()
```

5. To read data from the Object Store, call the `Read` method. You need to provide the key you used to store the object.

```
content = qb.ObjectStore.Read("sma_values_python")
```

6. Convert the data to a pandas object, and create a chart.

```
data = {}
for line in content.split('\n'):
    csv = line.split(',')
    if len(csv) > 1:
        data[csv[0]] = float(csv[1])

series = pd.Series(data, index=data.keys())
series.plot()
```



Preserve Insights Between Deployments

Follow these steps to use the Object Store to preserve the algorithm state across live deployments:

1. Create an algorithm that defines a storage key and adds insights to the Insight Manager .

```
class ObjectStoreChartingAlgorithm(QCAlgorithm):
    def Initialize(self):
        self.insight_key = f"{self.ProjectId}/insights"
        self.SetUniverseSelection(ManualUniverseSelectionModel([ Symbol.Create("SPY",
SecurityType.Equity, Market.USA) ]))
        self.SetAlpha(ConstantAlphaModel(InsightType.Price, InsightDirection.Up, timedelta(5), 0.025,
None))
```

2. At the top of the algorithm file, add the following imports:

```
from Newtonsoft.Json import JsonConvert
from System.Collections.Generic import List
```

   `Insight` objects are a C# objects, so you need the preceding C# libraries to serialize and deserialize them.

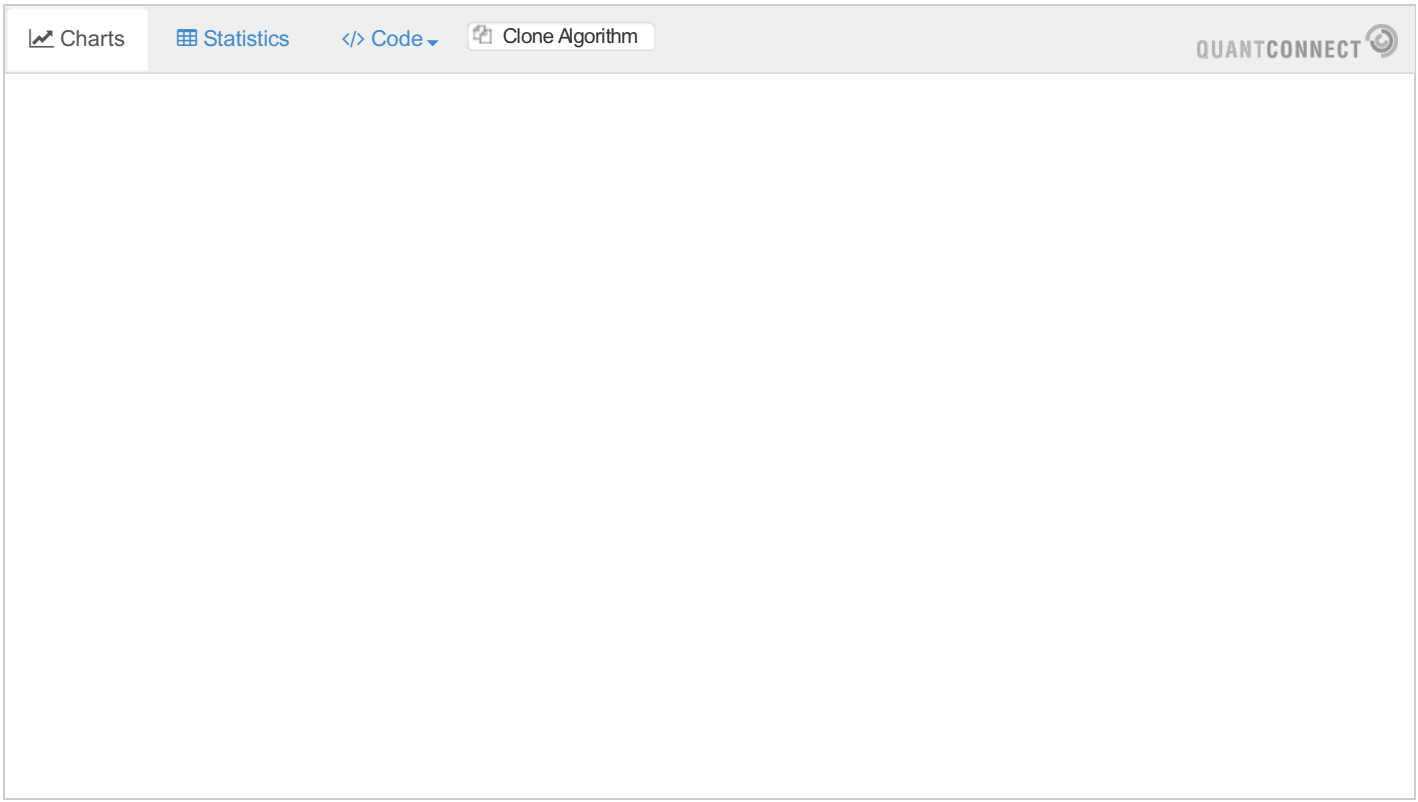
3. In the OnEndOfAlgorithm event handler of the algorithm, get the Insight objects and save them in the Object Store as a JSON object.

```
def OnEndOfAlgorithm(self):
    insights = self.Insights.GetInsights(lambda x: x.IsActive(self.UtcTime))
    content = ','.join([JsonConvert.SerializeObject(x) for x in insights])
    self.ObjectStore.Save(self.insight_key, f'[{content}]')
```

4. At the bottom of the `Initialize` method, read the Insight objects from the Object Store and add them to the Insight Manager .

```
if self.ObjectStore.ContainsKey(self.insight_key):
    insights = self.ObjectStore.ReadJson[List[Insight]](self.insight_key)
    self.Insights.AddRange(insights)
```

The following algorithm provides a full example of preserving the Insight state between deployments:



Live Trading Considerations

If you update the Object Store from outside of your live trading node and then try to read the new content from inside your live trading algorithm, you may not get the new content because the Object Store caches data to speed up execution. To ensure you get the latest value for a specific key in the Object Store, clear the cache and then read the data .

```
self.ObjectStore.Clear()
if self.ObjectStore.ContainsKey(key):
    value = self.ObjectStore.Read(key)
```

Parameters

Introduction

Parameters are project variables that your algorithm uses to define the value of internal variables like indicator arguments or the length of lookback windows. Parameters are stored outside of your algorithm code, but we inject the values of the parameters into your algorithm when you run a backtest, deploy a live algorithm, or launch an optimization job. To use parameters, set some parameters in your project and then load them into your algorithm.

Set Parameters

The process to set parameter values depends on the environment you use to write algorithms. See the tutorial in one of the following environments:

- Cloud Platform
- Local Platform
- LEAN CLI

Get Parameters

To get a parameter value into your algorithm, call the `GetParameter` method of the algorithm class.

```py
parameter_value = self.GetParameter("parameterName")
```

The `GetParameter` method returns a string by default. If you provide a default parameter value, the method returns the parameter value as the same data type as the default value. If there are no parameters in your project that match the name you pass to the method and you provide a default value to the method, it returns the default value. The following table describes the arguments the `GetParameter` method accepts:

Argument	Data Type	Description	Default Value
`name`	`str`	The name of the parameter to get	
`defaultValue`	`str/int/double`	The default value to return	`None`

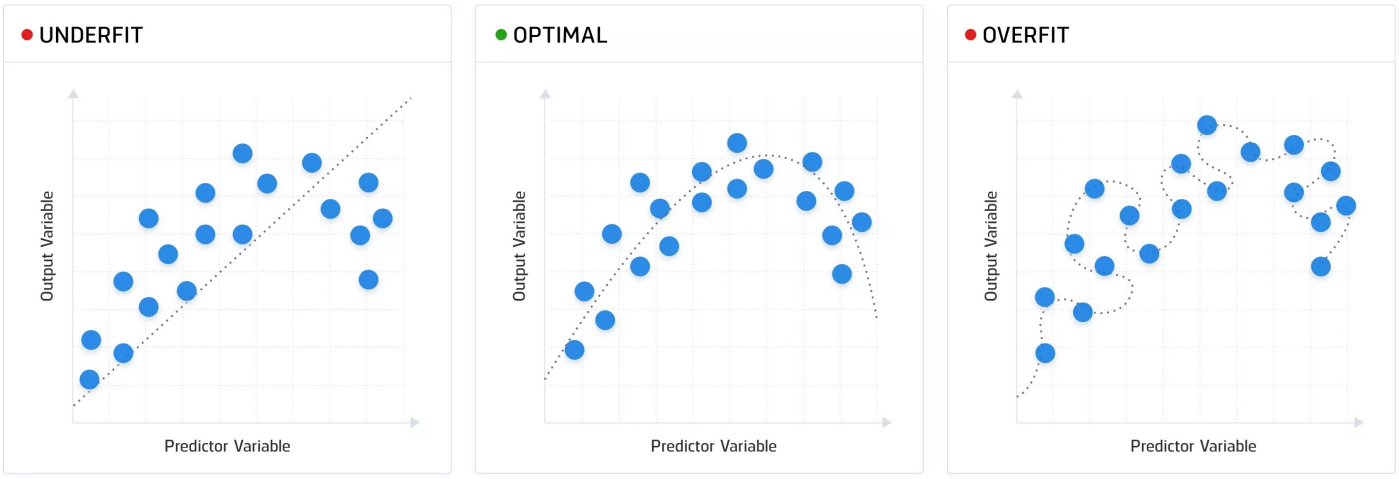
The following example algorithm gets the values of parameters of each data type:

```py
class ParameterizedAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        # Get the parameter value and return an integer
        int_parameter_value = self.GetParameter("<int_parameter_name>", 100)

        # Get the parameter value and return a double
        float_parameter_value = self.GetParameter("<float_parameter_name>", 0.95)

        # Get the parameter value as a string
        string_parameter_value = self.GetParameter("<parameter_name>", "default_string_value")

        # Cast it to an integer
        parameter_value = int(string_parameter_value)
```

The parameter values are sent to your algorithm when you deploy the algorithm, so it's not possible to change the parameter values while the algorithm runs.

Overfitting

Overfitting occurs when a function is fit too closely fit to a limited set of training data. Overfitting can occur in your trading algorithms if you have many parameters or select parameters values that worked very well in the past but are sensitive to small changes in their values. In these cases, your algorithm will likely be fine-tuned to fit the detail and noise of the historical data to the extent that it negatively impacts the live performance of your algorithm. The following image shows examples of underfit, optimally-fit, and overfit functions:



An algorithm that is dynamic and generalizes to new data is more likely to survive across different market conditions and apply to other markets.

Look-Ahead Bias

Look-ahead bias occurs when an algorithm makes decisions using data that would not have yet been available. For instance, in optimization jobs, you optimize a set of parameters over a historical backtesting period. After the optimizer finds the optimal parameter values, the backtest period becomes part of the in-sample data. If you run a backtest over the same period using the optimal parameters, look-ahead bias has seeped into your research. In reality, it would not be possible to know the optimal parameters during the testing period until after the testing period is over. To avoid issues with look-ahead bias, optimize on older historical data and test the optimal parameter values on recent historical data. Alternatively, apply walk forward optimization to optimize the parameters on smaller batches of history.

Live Trading Considerations

To update parameters in live mode, add a Schedule Event that downloads a remote file and uses its contents to update the parameter values.

```py
def Initialize(self):
    self.parameters = { }
    if self.LiveMode:
        def download_parameters():
            content = self.Download(url_to_remote_file)
            # Convert content to self.parameters

        self.Schedule.On(self.DateRules.EveryDay(), self.TimeRules.Every(timedelta(minutes=1)),
download_parameters)
```

Examples

The following example algorithm demonstrates loading parameter values :

```py
class ParameterizedAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2020, 1, 1)
        self.SetCash(100000)
        self.AddEquity("SPY")

        fast_period = self.GetParameter("ema-fast", 100)
        slow_period = self.GetParameter("ema-slow", 200)

        self._fast = self.EMA("SPY", fast_period)
        self._slow = self.EMA("SPY", slow_period)
```

Demonstration Algorithms
ParameterizedAlgorithm.py Python

Machine Learning

Machine Learning

Key Concepts

Introduction

Machine learning is a field of study that combines statistics and computer science to build intelligent systems that predict outcomes. You can use machine learning techniques in your trading strategies.

Supported Libraries

LEAN supports several machine learning libraries. You can import these packages and use them in your algorithms.

Name	Version	Language	Import Statement	Exam
TensorFlow	2.11.0	Python	`import tensorflow`	
SciKit Learn	1.2.1	Python	`import sklearn`	
Py Torch	1.13.1	Python	`import torch`	
Keras	2.11.0	Python	`import keras`	
gplearn	0.4.2	Python	`import gplearn`	
hmmlearn	0.2.8	Python	`import hmmlearn`	
tsfresh	0.20.0	Python	`import tsfresh`	
Stable-Baselines3	1.7.0	Python	`from stable_baselines3 import *`	
fastai	2.7.11	Python	`import fastai`	
Deap	1.3.3	Python	`import deap`	
XGBoost	1.7.4	Python	`import xgboost`	
mlfinlab	1.6.0	Python	`import mlfinlab`	
Accord	3.6.0	C#	`using Accord.MachineLearning;`	

Add New Libraries

To request a new library, contact us . We will add the library to the queue for review and deployment. Since the libraries

run on our servers, we need to ensure they are secure and won't cause harm. The process of adding new libraries takes 2-4 weeks to complete. View the list of libraries currently under review on the Issues list of the Lean GitHub repository .

Save Models

After you train a model, you can save it into the Object Store . In QuantConnect Cloud, we back up your Object Store data on QuantConnect servers. In local algorithms, your local machine saves the Object Store data. If you save models in live algorithms, save them at the end of the training method so you can access the trained model again if your algorithm stops executing. If you save models in backtests, save them during the `OnEndOfAlgorithm` event handler so that saving multiple times doesn't slow down your backtest.

To view examples of storing library-specific models, see Popular Libraries .

Load Models

You can load machine learning models from the Object Store or a custom data file like pickle. If you load models from the Object Store, before you load the model into your algorithm, in the `Initialize` method, check if the Object Store already contains the model. To avoid look-ahead bias in backtests, don't train your model on the same data you use to test the model.

Machine Learning

Training Models

Introduction

Algorithms usually must process each timeslice within 10 minutes, but the `Train` method allows you to increase this time to train machine learning models. The length of time you can train depends on your training quotas.

Train Models

To train models immediately, call the `Train` method and pass in the name of your training method.

```
self.Train(self.MyMethod)
```

Immediate training is most useful for training your model when you first deploy your strategy to production or when the model's performance begins to degrade.

Schedule Training Sessions

You can schedule model training sessions in a similar way to a Scheduled Event. To schedule a training session do this, pass in a DateRules and TimeRules argument to the `Train` method.

```
# Set TrainingMethod to be executed at 8:00 am every Sunday
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8,0), self.MyTrainingMethod)
```

We recommend you schedule your training sessions for when the market is closed to get the best compute allocation. While the market is open, your CPU is occupied with processing incoming tick data and handling other LEAN events.

Training Quotas

Training resources are allocated with a leaky bucket algorithm where you can use a maximum of n-minutes in a single training session and the number of available minutes refills over time. This design gives you burst allocations when you need them and recharges the allowance to prepare for the next training.

Cloud Quotas

If you execute algorithms in QuantConnect Cloud, see Training Quotas for more information about the training quotas.

Local Quotas

If you execute algorithms locally, the following table shows the default settings for the leaky bucket algorithm:

Setting	Value
Capacity (minutes)	120
Time interval (minutes)	1440
Refill amount (minutes per time interval)	Capacity / 7

To allow virtually unlimited training for local algorithms, add the following key-value pairs to your **Lean / Launcher / config.json** file:

```
"scheduled-event-leaky-bucket-capacity" : 99999999,
"scheduled-event-leaky-bucket-time-interval-minutes" : 1,
"scheduled-event-leaky-bucket-refill-amount": 999999,
```

Check Model Readiness

In backtests, the `Train` method is synchronous, so it blocks your algorithm execution while the model trains. In live trading, the `Train` method is asynchronous, so ensure your model is trained before you continue the algorithm execution. Training occurs on a separate thread, so set a boolean flag to notify your algorithm of the model state. A **semaphore** is a thread-safe flag you can use to synchronize program operations across different threads.

PY

```
class SemaphoreTrainingAlgorithm(QCAlgorithm):

    # Model Object
    model = None
    # Model State Flag
    model_is_training = False

    def Initialize(self) -> None:
        self.Train(self.my_training_method)

    def my_training_method(self) -> None:
        self.model_is_training = True
        # Perform Work
        self.model_is_training = False

    def OnData(self, slice: Slice) -> None:
        # Do not use model while it is being trained.
        if self.model_is_training:
            return

        # Once training is complete; use the model safely.
        result = self.model.Predict()
```

Examples

Demonstration Algorithms
TrainingExampleAlgorithm.py Python

Machine Learning

Popular Libraries

These are examples of using some of the most common machine learning libraries in an algorithm. Click one to learn more.

GPlearn

Hmmlearn

Keras

MlFinLab

PyTorch

Scikit-Learn

Stable Baselines

Tensorflow

Tslearn

XGBoost

Aesera

Popular Libraries

GPlearn

Introduction

This page explains how to build, train, deploy and store `GPlearn` models.

Import Libraries

Import the `gplearn` and `joblib` libraries.

```
PY
from AlgorithmImports import *
from gplearn.genetic import SymbolicRegressor, SymbolicTransformer
import joblib
```

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `GPLearn` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a genetic programming feature transformation model and a genetic programming regression prediction model using the following features and labels:

Data Category	Description
Features	Daily percent change of the close price of the SPY over the last 5 days
Labels	Daily percent return of the SPY over the next day

The following image shows the time difference between the features and labels:

Follow these steps to create a method to build the model:

1. Declare a set of functions to use for feature engineering.

```
function_set = ['add', 'sub', 'mul', 'div',
                'sqrt', 'log', 'abs', 'neg', 'inv',
                'max', 'min']
```

2. Call the `SymbolicTransformer` constructor with the preceding set of functions and then save it as a class variable.

```
self.gp_transformer = SymbolicTransformer(function_set=function_set)
```

3. Call the `SymbolicRegressor` constructor to instantiate the regression model.

```
self.model = SymbolicRegressor()
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
training_length = 252*2
self.training_data = RollingWindow[float](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar.Close)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
PY
def get_features_and_labels(self, n_steps=5):
    training_df = list(self.training_data)[::-1]
    daily_pct_change = ((np.roll(training_df, -1) - training_df) / training_df)[:-1]

    features = []
    labels = []
    for i in range(len(daily_pct_change)-n_steps):
        features.append(daily_pct_change[i:i+n_steps])
        labels.append(daily_pct_change[i+n_steps])
    features = np.array(features)
    labels = np.array(labels)

    return features, labels

def my_training_method(self):
    features, labels = self.get_features_and_labels()

    # Feature engineering
    self.gp_transformer.fit(features, labels)
    gp_features = self.gp_transformer.transform(features)
    new_features = np.hstack((features, gp_features))

    # Fit the regression model with transformed and raw features.
    self.model.fit(new_features, labels)
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
PY
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
PY
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current close price to the `RollingWindow` that holds the training data.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol].Close)
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
features, _ = self.get_features_and_labels()

# Get transformed features
gp_features = self.gp_transformer.transform(features)
new_features = np.hstack((features, gp_features))

# Get next prediction
prediction = self.model.predict(new_features)
prediction = float(prediction.flatten()[-1])
```

You can use the label prediction to place orders.

```
if prediction > 0:
    self.SetHoldings(self.symbol, 1)
elif prediction < 0:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `GPLearn` models into the ObjectStore :

1. Set the key names you want to store the models under in the ObjectStore.

   ```
   transformer_model_key = "transformer"
   regressor_model_key = "regressor"
   ```

2. Call the `GetFilePath` method with the keys.

   ```
   transformer_file_name = self.ObjectStore.GetFilePath(transformer_model_key)
   regressor_file_name = self.ObjectStore.GetFilePath(regressor_model_key)
   ```

   This method returns the file paths where the models will be stored.

3. Call the `dump` method the file paths.

   ```
   joblib.dump(self.gp_transformer, transformer_file_name)
   joblib.dump(self.model, regressor_file_name)
   ```

   If you dump the models using the `joblib` module before you save the models, you don't need to retrain the models.

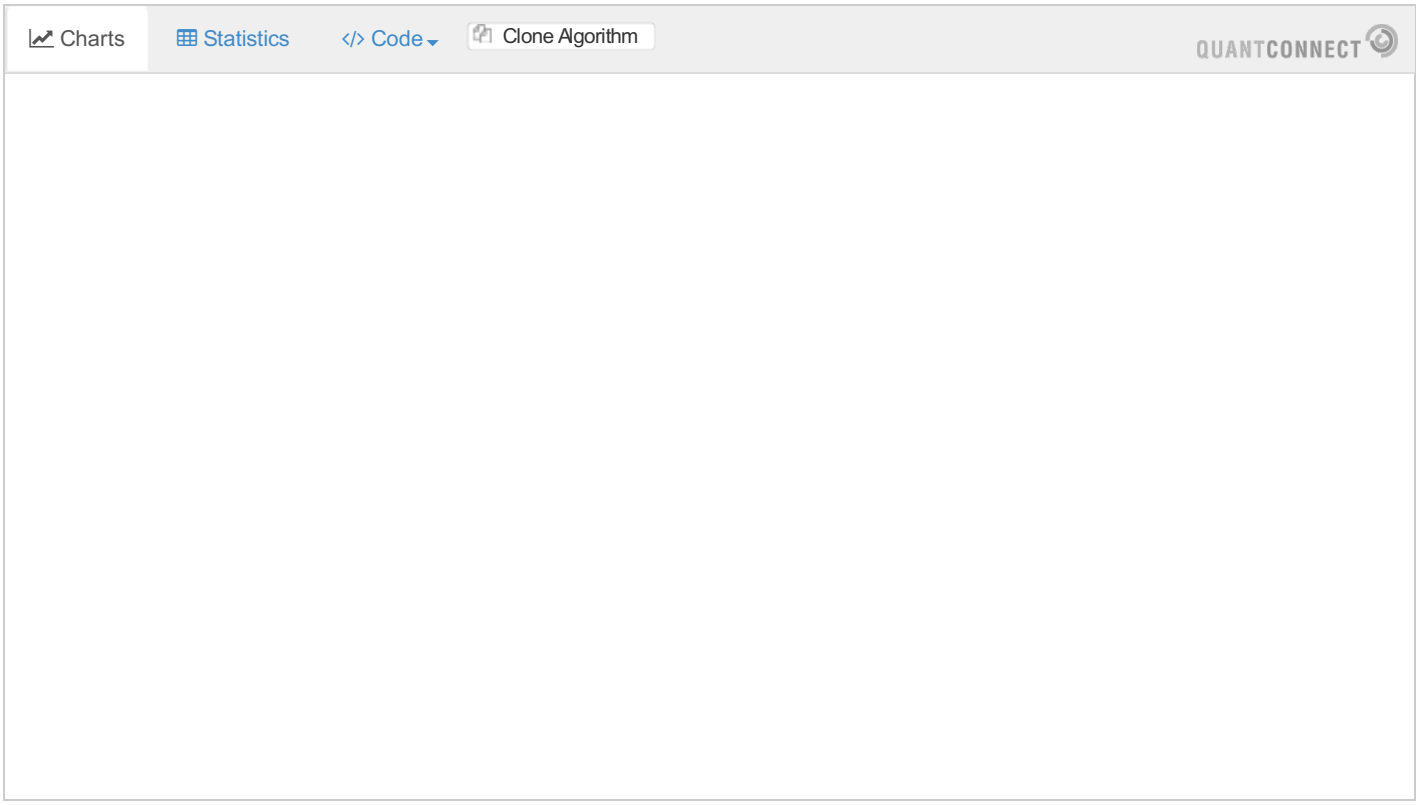
Load Models

You can load and trade with pre-trained `GPLearn` models that you saved in the ObjectStore. To load a `GPLearn` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(transformer_model_key) and
self.ObjectStore.ContainsKey(regressor_model_key):
        transformer_file_name = self.ObjectStore.GetFilePath(transformer_model_key)
        regressor_file_name = self.ObjectStore.GetFilePath(regressor_model_key)
        self.gp_transformer = joblib.load(transformer_file_name)
        self.model = joblib.load(regressor_file_name)
```
PY

The `ContainsKey` method returns a boolean that represents if `transformer_model_key` and `regressor_model_key` are in the ObjectStore. If the ObjectStore does not contain the keys, save the model using them before you proceed.

Clone Example Algorithm



Popular Libraries

Hmmlearn

Introduction

This page explains how to build, train, deploy and store `Hmmlearn` models.

Import Libraries

Import the `hmmlearn` and `joblib` libraries.

```
PY
from AlgorithmImports import *
from hmmlearn import hmm
import joblib
```

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `hmmlearn` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, assume the market has only 2 regimes and the market returns follow a Gaussian distribution. Therefore, create a 2-component Hidden Markov Model with Gaussian emissions, which is equivalent to a Gaussian mixture model with 2 means.

To build the model, call the `GaussianHMM` constructor with the number of components, a covariance type, and the number of iterations:

```
PY
self.model = hmm.GaussianHMM(n_components=2, covariance_type="full", n_iter=100)
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
training_length = 252*2
self.training_data = RollingWindow[float](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar.Close)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
def get_features(self):
    training_df = np.array(list(self.training_data)[::-1])
    daily_pct_change = (np.roll(training_df, 1) - training_df) / training_df

    return daily_pct_change[1:].reshape(-1, 1)

def my_training_method(self):
    features = self.get_features()
    self.model.fit(features)
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current close price to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol].Close)
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
new_feature = self.get_features()
prediction = self.model.predict(new_feature)
prediction = float(prediction[-1])
```

You can use the label prediction to place orders.

```
if prediction == 1:
    self.SetHoldings(self.symbol, 1)
else:
    self.Liquidate(self.symbol)
```

Save Models

Follow these steps to save `hmmlearn` models into the ObjectStore :

1. Set the key name you want to store the model under in the ObjectStore.

   ```
   model_key = "model.hmm"
   ```

2. Call the `GetFilePath` method with the key.

   ```
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   ```
   joblib.dump(self.model, file_name)
   ```

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

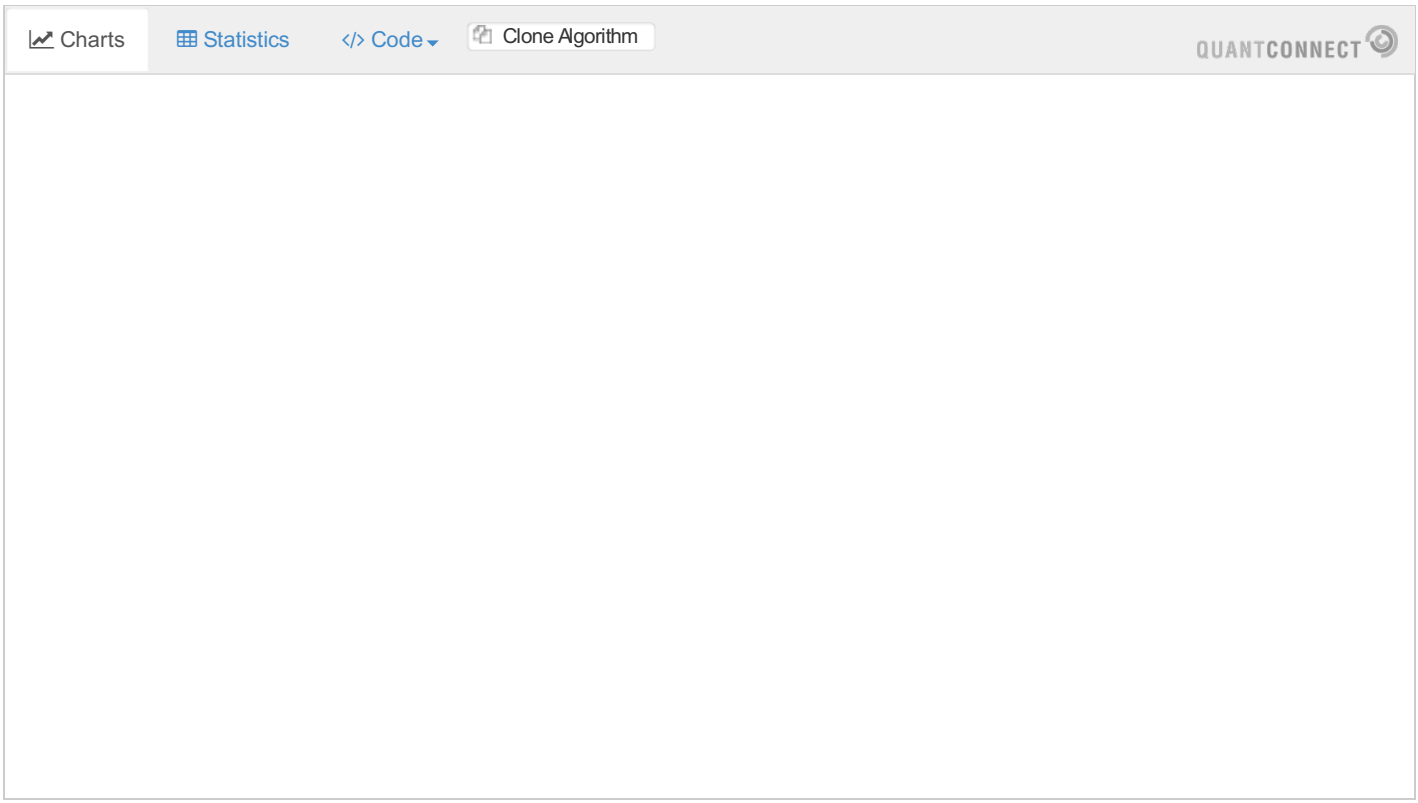
Load Models

You can load and trade with pre-trained `hmmlearn` models that you saved in the ObjectStore. To load a `hmmlearn` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = joblib.load(file_name)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Popular Libraries

Keras

Introduction

This page explains how to build, train, deploy and store `Keras` models.

Import Libraries

Import the `keras` libraries.

```
PY
from AlgorithmImports import *
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `keras` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a neural-network regression model that uses the following features and labels:

Data Category	Description
Features	Daily percent change of the open, high, low, close, and volume of the SPY over the last 5 days
Labels	Daily percent return of the SPY over the next day

The following image shows the time difference between the features and labels:

Follow the below steps to build the model:

1. In the `Initialize` method, create a `Sequential` object with several layers.

```
PY
self.model = Sequential([Dense(10, input_shape=(5,5), activation='relu'),
                         Dense(10, activation='relu'),
                         Flatten(),
                         Dense(1)])
```

   Set the `input_shape` of the first layer to `(5, 5)` because each sample contains the percent change of 5 factors (percent change of the open, high, low, close, and volume) over the previous 5 days. Call the `Flatten` constructor because the input is 2-dimensional but the output is just a single value.

2. Call the `compile` method with a loss function, an optimizer, and a list of metrics to monitor.

```
PY
self.model.compile(loss='mse',
                   optimizer=Adam(),
                   metrics=['mae', 'mse'])
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
training_length = 252*2
self.training_data = RollingWindow[TradeBar](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
def get_features_and_labels(self, n_steps=5):
    training_df = self.PandasConverter.GetDataFrame[TradeBar](list(self.training_data)[::-1])
    daily_pct_change = training_df.pct_change().dropna()

    features = []
    labels = []
    for i in range(len(daily_pct_change)-n_steps):
        features.append(daily_pct_change.iloc[i:i+n_steps].values)
        labels.append(daily_pct_change['close'].iloc[i+n_steps])
    features = np.array(features)
    labels = np.array(labels)

    return features, labels

def my_training_method(self):
    features, labels = self.get_features_and_labels()
    self.model.fit(features, labels, epochs=5)
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol])
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict`

method.

```
features, _ = self.get_features_and_labels()
features = features[-1].reshape(1, 5, 5)
prediction = float(self.model.predict(features)[-1])
```

You can use the label prediction to place orders.

```
if prediction > 0:
    self.SetHoldings(self.symbol, 1)
elif prediction < 0:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `keras` models into the ObjectStore :

1. Set the key name you want to store the model under in the ObjectStore.

   ```
   model_key = "model"
   ```

2. Call the `GetFilePath` method with the key.

   ```
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `save` method with the file path.

   ```
   self.model.save(file_name)
   ```

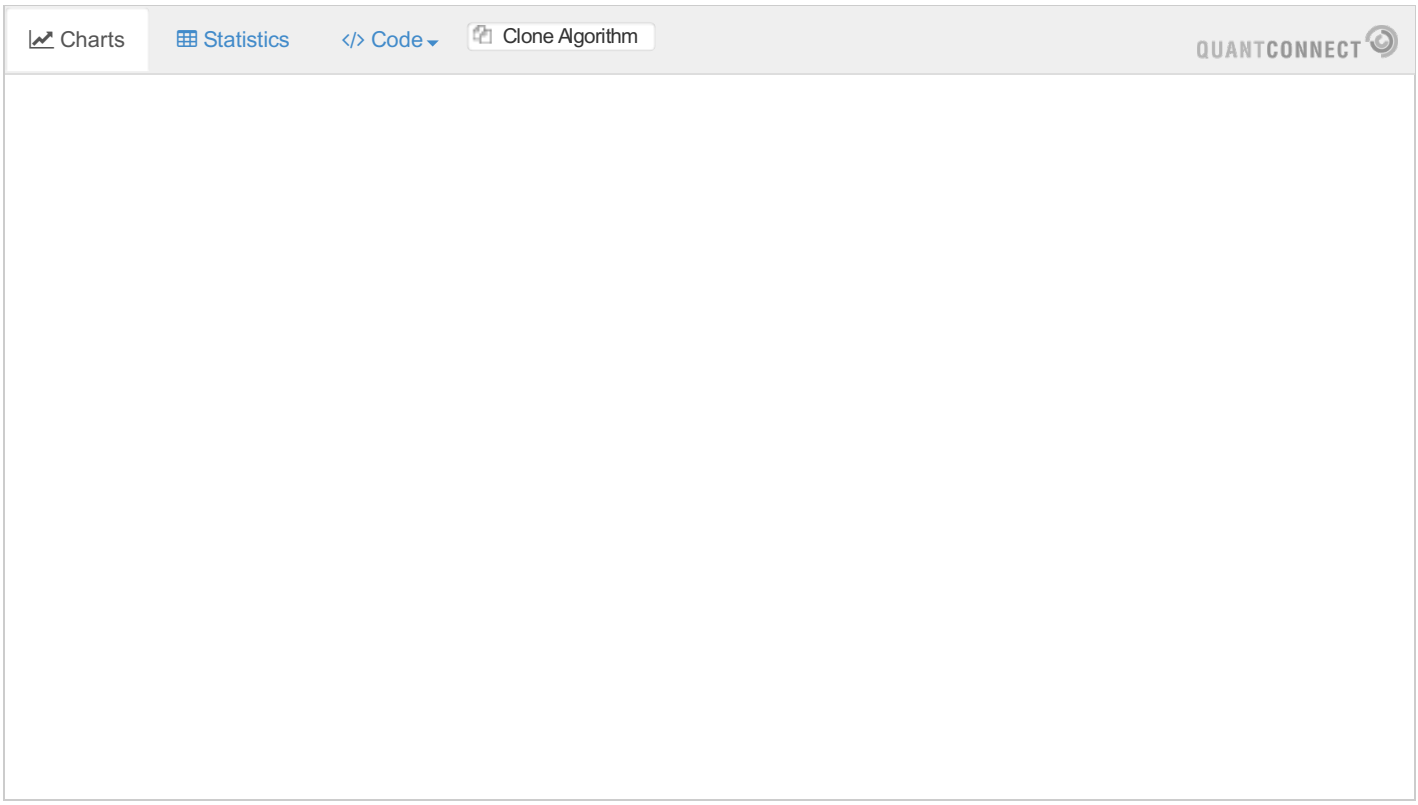
Load Models

You can load and trade with pre-trained `keras` models that you saved in the ObjectStore. To load a `keras` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load_model` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = load_model(file_name)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Popular Libraries

MlFinLab

Introduction

MlFinLab is a collection of production-ready algorithms (from the best journals and graduate-level textbooks), packed into a python library that enables portfolio managers and traders who want to leverage the power of machine learning by providing reproducible, interpretable, and easy to use tools. It covers every step of the machine learning strategy creation starting from data structures generation and finishing with backtest statistics.

The mission of Hudson and Thames is to promote the scientific method within investment management by codifying frameworks, algorithms, and best practices to build the world's central repository of ready to use implementations and intellectual property. For more information, see the Hudson and Thames website . The Hudson and Thames environment, which includes the MlFinLab library, is currently only available in QuantConnect Cloud.

Import Libraries

Follow these steps to import the `mlfinlab` library:

1. Open a project .
2. In the Project panel, click the **LEAN Environment** field and then click **Hudson & Thames** from the drop-down menu.
3. Open one of the code files in your project.
4. At the top of the code file, add the following snippet:

```
PY
import ht_auth
ht_auth.SetToken("<mlfinlab_api_key>")
import mlfinlab as ml
```

MlFinLab costs £100 (+VAT) per month, per user. Follow these steps to get your API key:

1. Create a profile on the Hudson & Thames website and log in.
2. On the dashboard page under MlFinLab, click **Buy** .
3. On the MlFinLab page, click **Buy** .
4. In the Terms of Use window, accept the terms of use and then click **Purchase** .
5. In the Purchase MlFinLab window, enter your payment card details and then click **Subscribe Now** .
6. Wait for the success window to display.
7. On the dashboard page, under MlFinLab, click **View** .
8. On the MlFinLab page, click **Copy API Key** .

Financial Data Structures

Transform unstructured data sets into structured tick, volume, and dollar bars as well as the less common information-

driven bars. With MlFinLab you can even generate bars on the go with our online data structures framework. For more information about data structures, see Standard Bars in the MlFinLab documentation.

Labeling Techniques

MlFinLab provides a comprehensive list of labeling techniques, including the following: Raw Returns, Fixed Horizon, Triple-Barrier & Meta-Labeling, and many more. For more information about labeling, see Labeling in the MlFinLab documentation.

Feature Engineering

This section of MlFinLab contains several important tools to manipulate, select and transform raw data into useful features. Feature engineering techniques you can use include: Fractionally Differentiated Features, Structural Breaks, Volatility Estimators, and even Automatic Feature Extraction. For more information about feature engineering, see Feature Importance in the MlFinLab documentation.

Bet Sizing

This section will allow you to optimally size your bets to improve returns or limit downside metrics in alignment with your risk preferences. Methods include Kelly Criterion, EF3M and dynamic bet sizes. For more information about bet sizing, see Bet Sizing in the MlFinLab documentation.

Codependence Measures

Improve your strategies by taking into account measures of codependence between assets in MlFinLab. Choose between correlation-based, information theory and copula-based metrics. For more information about codependence measures, see Measures of Codependence in the MlFinLab documentation.

Generate Sythetic Data

Generate data that simulate important events for example, flash crashes, world-wide economic crises, global pandemics, etc. to assess if an algorithm will fare well for any event. Having an abundance of realistic financial data has never been easier with MlFinLab. Examples of financial data that can be generated include stock prices, stock returns, correlation matrices, retail banking data, and all kinds of market microstructure data. For more information about synthetic data, see Synthetic Data Generation in the MlFinLab documentation.

Clustering Techniques

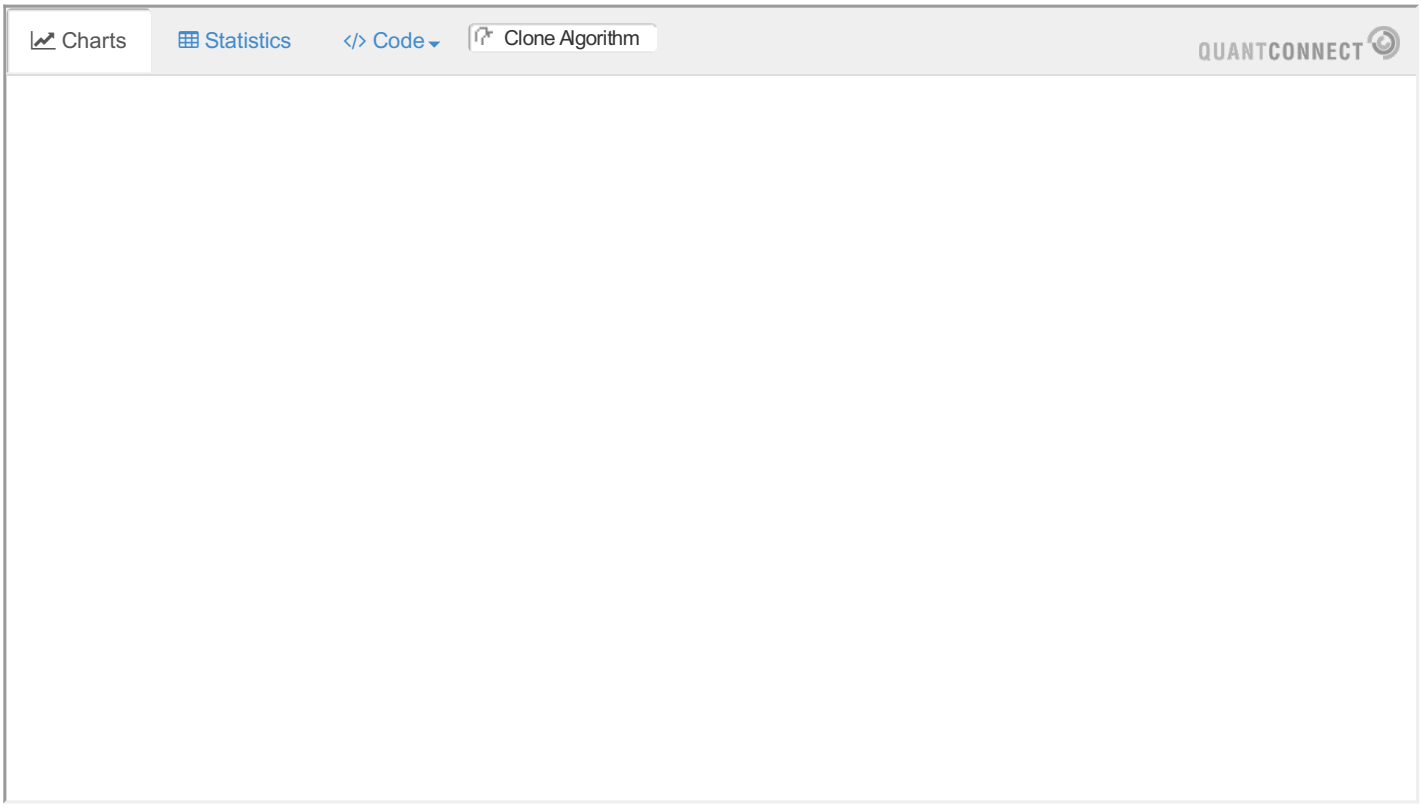
Discover a wide range of clustering techniques at your fingertips to improve the credibility of your investment strategy. MlFinLab offers both hierarchical clustering techniques and the ability to determine the optimal number of clusters. For more information about clustering techniques, see Clustering in the MlFinLab documentation.

Networks Modeling

Creates beautiful and informative visualizations of financial data, using network theory to describe complex systems such as financial markets. Use graphs such as a Minimum Spanning Tree, creating a mini Flask server using Plotly's Dash to display the interactive graphs with MlFinLab. For more information about networks modeling, see Networks in the MlFinLab documentation.

Example Algorithm

The following example algorithm demonstrates how to use the `mlfinlab` library. The algorithm uses the `trend_scanning_labels` method to determine the trend direction of the Bitcoin-USD pair. If Bitcoin is in an uptrend, the algorithm allocations 100% of the portfolio to Bitcoin. Otherwise, it holds USD. The algorithm achieves a 1.79 Sharpe ratio, outperforming Bitcoin buy-and-hold, which achieves a 1.26 Sharpe ratio over the same time period.



Support

After you purchase MlFinLab, you get access to the Hudson and Thames Slack Community, where you and other quants can answer questions. Hudson and Thames also provide support under consulting.

FAQs

Note the following frequently asked questsions.

How do I install MlFinLab on QuantConnect?

To use MlFinLab on QuantConnect, see Import Libraries .

How do I install MlFinLab on my local machine?

To use MlFinLab on your local machine, see the Client Documentation that Hudson & Thames provides to avoid dependency issues. You receive the MlFinLab Client documentation after your purchase is successful.

Are the Jupyter Notebooks downloadable?

Yes, you can access the notebooks via the Client Documentation and download them.

Does Hudson & Thames provide support?

Yes, they provide support under consulting and you can ask their community on Slack if you have a question.

Popular Libraries

PyTorch

Introduction

This page explains how to build, train, deploy and store `PyTorch` models.

Import Libraries

Import the `torch` and `joblib` libraries.

```
PY
from AlgorithmImports import *
import torch
from torch import nn
import joblib
```

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `torch` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a neural-network regression model that uses the following features and labels:

Data Category	Description
Features	The last 5 closing prices.
Labels	The following day's closing price

The following image shows the time difference between the features and labels:

Follow these steps to create a method to build the model:

1. Define a subclass of `nn.Module` to be the model.

   In this example, use the ReLU activation function for each layer.

```
class NeuralNetwork(nn.Module):
    # Model Structure
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(5, 5),    # input size, output size of the layer
            nn.ReLU(),          # Relu non-linear transformation
            nn.Linear(5, 5),
            nn.ReLU(),
            nn.Linear(5, 1),    # Output size = 1 for regression
        )

    # Feed-forward training/prediction
    def forward(self, x):
        x = torch.from_numpy(x).float()   # Convert to tensor in type float
        result = self.linear_relu_stack(x)
        return result
```

2. Create an instance of the model and set its configuration to train on the GPU if it's available.

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
self.model = NeuralNetwork().to(device)
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
training_length = 252*2
self.training_data = RollingWindow[float](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar.Close)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
def get_features_and_labels(self, n_steps=5):
    close_prices = list(self.training_data)[::-1]

    features = []
    labels = []
    for i in range(len(close_prices)-n_steps):
        features.append(close_prices[i:i+n_steps])
        labels.append(close_prices[i+n_steps])
    features = np.array(features)
    labels = np.array(labels)

    return features, labels

def my_training_method(self):
    features, labels = self.get_features_and_labels()

    # Set the loss and optimization functions
    # In this example, use the mean squared error as the loss function and stochastic gradient descent as
the optimizer
    loss_fn = nn.MSELoss()
    learning_rate = 0.001
    optimizer = torch.optim.SGD(self.model.parameters(), lr=learning_rate)

    # Create a for-loop to train for preset number of epoch
    epochs = 5
    for t in range(epochs):
        # Create a for-loop to fit the model per batch
        for batch, (feature, label) in enumerate(zip(features, labels)):
            # Compute prediction and loss
            pred = self.model(feature)
            real = torch.from_numpy(np.array(label).flatten()).float()
            loss = loss_fn(pred, real)

            # Perform backpropagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
PY
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol].Close)
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and pass it to the model.

```
PY
features, __ = self.get_features_and_labels()
prediction = self.model(features[-1].reshape(1, -1))
prediction = float(prediction.detach().numpy()[-1])
```

You can use the label prediction to place orders.

```
PY
if prediction > slice[self.symbol].Price:
    self.SetHoldings(self.symbol, 1)
elif prediction < slice[self.symbol].Price:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `PyTorch` models into the ObjectStore :

1. Set the key name of the model to be stored in the ObjectStore.

   ```
   PY
   model_key = "model"
   ```

2. Call the `GetFilePath` method with the key.

   ```
   PY
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   ```
   PY
   joblib.dump(self.model, file_name)
   ```

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

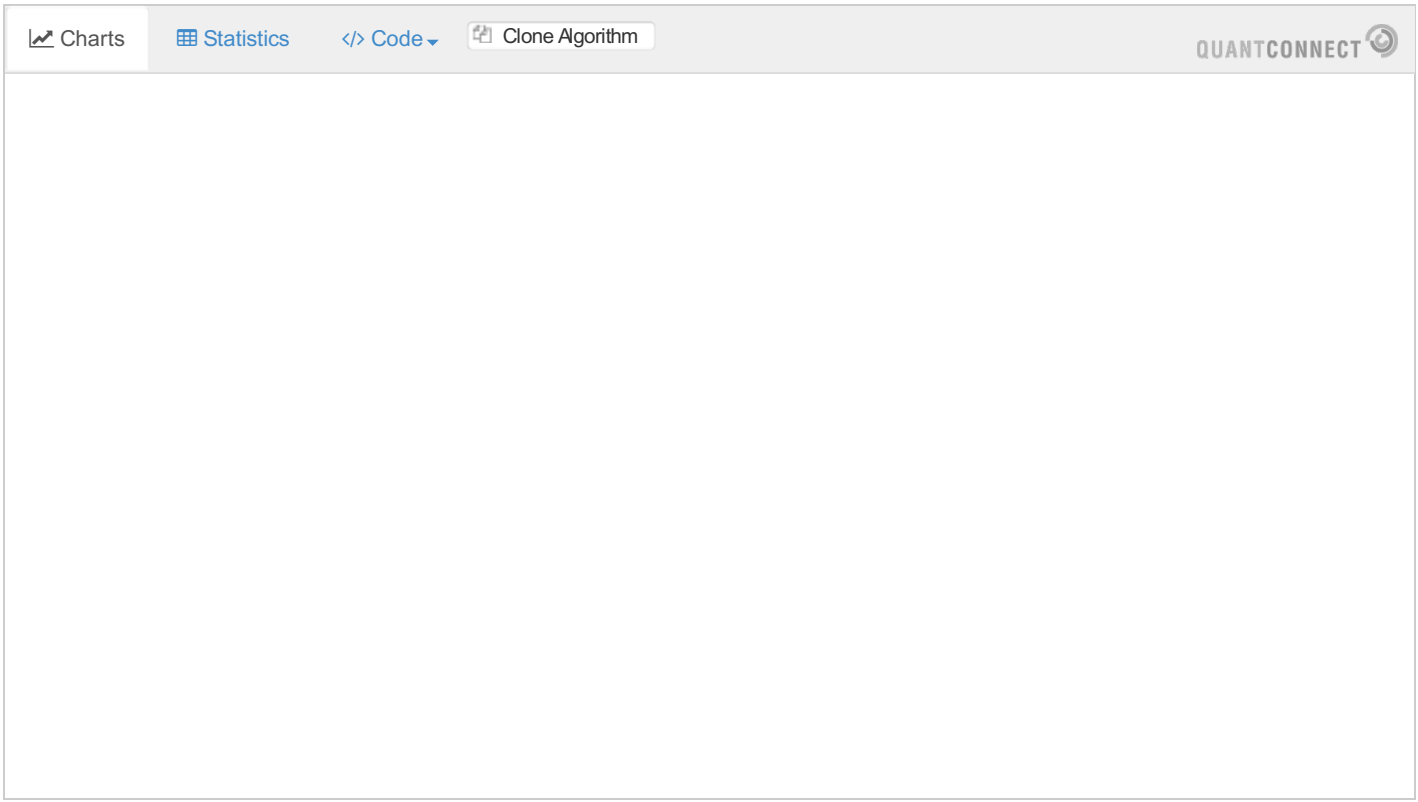
Load Models

You can load and trade with pre-trained `PyTorch` models that you saved in the ObjectStore. To load a `PyTorch` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

```py
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = joblib.load(file_name)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm



Popular Libraries

Scikit-Learn

Introduction

This page explains how to build, train, deploy and store `Scikit-Learn` models.

Import Libraries

Import the `sklearn` and `joblib` libraries.

```
PY
from AlgorithmImports import *
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
import joblib
```

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `sklearn` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a support vector regression prediction model that uses the following features and labels:

Data Category	Description
Features	Daily percent change of the open, high, low, close, and volume of the SPY over the last 5 days
Labels	Daily percent return of the SPY over the next day

The following image shows the time difference between the features and labels:

To build the model, call the `GridSearchCV` constructor with the SVR model, the parameter grid, a scoring method, the number of cross-validation folds:

```
PY
param_grid = {'C': [.05, .1, .5, 1, 5, 10],
              'epsilon': [0.001, 0.005, 0.01, 0.05, 0.1],
              'gamma': ['auto', 'scale']}
self.model = GridSearchCV(SVR(), param_grid, scoring='neg_mean_squared_error', cv=5)
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
PY
training_length = 252*2
self.training_data = RollingWindow[TradeBar](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
def get_features_and_labels(self, n_steps=5):
    training_df = self.PandasConverter.GetDataFrame[TradeBar](list(self.training_data)[::-1])
    daily_pct_change = training_df.pct_change().dropna()

    features = []
    labels = []
    for i in range(len(daily_pct_change)-n_steps):
        features.append(daily_pct_change.iloc[i:i+n_steps].values.flatten())
        labels.append(daily_pct_change['close'].iloc[i+n_steps])
    features = np.array(features)
    labels = np.array(labels)

    return features, labels

def my_training_method(self):
    features, labels = self.get_features_and_labels()
    self.model = self.model.fit(features, labels).best_estimator_
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol])
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
features, _ = self.get_features_and_labels()
prediction = self.model.predict(features[-1].reshape(1, -1))
prediction = float(prediction)
```

You can use the label prediction to place orders.

```py
if prediction > 0:
    self.SetHoldings(self.symbol, 1)
elif prediction < 0:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `sklearn` models into the ObjectStore :

1. Set the key name you want to store the model under in the ObjectStore.

   ```py
   model_key = "model"
   ```

2. Call the `GetFilePath` method with the key.

   ```py
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   ```py
   joblib.dump(self.model, file_name)
   ```

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

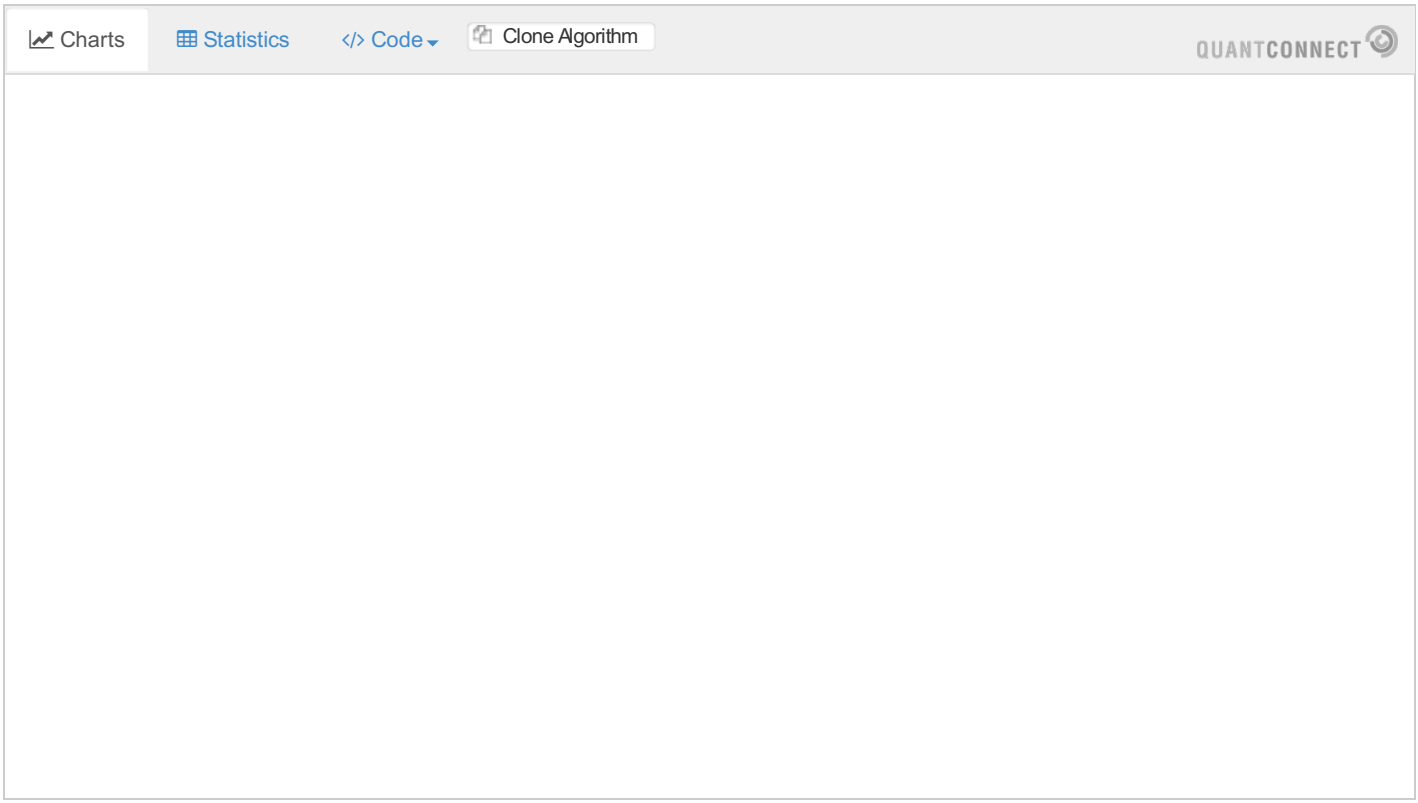
Load Models

You can load and trade with pre-trained `sklearn` models that you saved in the ObjectStore. To load a `sklearn` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

```py
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = joblib.load(file_name)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Popular Libraries

Stable Baselines

Introduction

This page explains how to build, train, deploy and store `stable baselines 3` models.

Import Libraries

Import the `gym` and `stable_baselines3` libraries.

```
PY
from AlgorithmImports import *
import gym
from stable_baselines3 import DQN
```

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `stable_baselines` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, create a `gym` environment to initialize the training environment, agent, and reward. Then, create a reinforcement learning model by a single-asset deep Q-network learning algorithm using the following observations and rewards:

Data Category	Description
Observations	The 5-day open, high, low, close, and volume (OHLCV) of the SPY
Rewards	Maximum portfolio return

Follow these steps to create a method to build the model:

1. Create a custom `gym` environment class.

   In this example, create a custom environment with the previous 5 OHLCV log-return data as observation and the highest portfolio value as reward.

```
PY
class TradingEnv(gym.Env):
    FLAT = 0
    LONG = 1
    SHORT = 2

    def __init__(self, ohlcv, ret):
        super(TradingEnv, self).__init__()

        self.ohlcv = ohlcv
        self.ret = ret
        self.trading_cost = 0.01
        self.reward = 1

        # The number of step the training has taken, starts at 5 since we're using the previous 5
data for observation.
        self.current_step = 5
        # The last action
        self.last_action = 0

        # Define action and observation space
        # Example when using discrete actions, we have 3: LONG, SHORT and FLAT.
        n_actions = 3
        self.action_space = gym.spaces.Discrete(n_actions)
        # The observation will be the coordinate of the agent, shape for (5 previous data poionts,
OHLCV)
        self.observation_space = gym.spaces.Box(low=-2, high=2, shape=(5, 5, 5), dtype=np.float64)

    def reset(self):
        # Reset the number of step the training has taken
        self.current_step = 5
        # Reset the last action
        self.last_action = 0
        # must return np.array type
        return self.ohlcv[self.current_step-5:self.current_step].astype(np.float32)

    def step(self, action):
        if action == self.LONG:
            self.reward *= 1 + self.ret[self.current_step] - (self.trading_cost if self.last_action
!= action else 0)
        elif action == self.SHORT:
            self.reward *= 1 + -1 * self.ret[self.current_step] - (self.trading_cost if
self.last_action != action else 0)
        elif action == self.FLAT:
                self.reward *= 1 - (self.trading_cost if self.last_action != action else 0)
        else:
            raise ValueError("Received invalid action={} which is not part of the action
space".format(action))

        self.last_action = action
        self.current_step += 1

        # Have we iterate all data points?
        done = (self.current_step == self.ret.shape[0]-1)

        # Reward as return
        return self.ohlcv[self.current_step-5:self.current_step].astype(np.float32), self.reward,
done, {}
```

2. Get the processed training data.

```
PY
obs, rewards = self.get_observations_and_rewards()
```

3. Initialize the environment with the observations and results.

```
PY
self.env = TradingEnv(obs, rewards)
```

4. Call the `DQN` constructor with the learning policy and the `gym` environment.

```
PY
self.model = DQN(MlpPolicy, env)
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
PY
training_length = 252*2
self.training_data = RollingWindow[TradeBar](training_length)
history = self.History[TradeBar](self.spy, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
PY
def get_observations_and_rewards(self, n_step=5):
    training_df = self.PandasConverter.GetDataFrame[TradeBar](list(self.training_data)[::-1])
    daily_pct_change = training_df['close'].pct_change().dropna()

    obs = []
    rewards = []
    for i in range(len(daily_pct_change)-n_step):
        obs.append(training_df.iloc[i:i+n_step].values)
        rewards.append(float(daily_pct_change.iloc[i+n_step]))
    obs = np.array(obs)
    rewards = np.array(rewards)

    return obs, rewards

def my_training_method(self):
    obs, rewards = self.get_observations_and_rewards()
    self.env = TradingEnv(obs, rewards)
    self.model = DQN("MlpPolicy", self.env)
    self.model.learn(total_timesteps=500)
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
PY
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol])
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
features, _ = self.get_observations_and_rewards()
action, _ = self.model.predict(features[-5:], deterministic=True)
_, _, _, _ = self.env.step(action)
```

You can use the label prediction to place orders.

```
if action == 0:
    self.Liquidate(self.spy)
elif action == 1:
    self.SetHoldings(self.spy, 1)
elif action == 2:
    self.SetHoldings(self.spy, -1)
```

Save Models

Follow these steps to save `stable_baselines` models into the ObjectStore :

1. Set the key name of the model to be stored in the ObjectStore.

   ```
   model_key = "model"
   ```

2. Call the `GetFilePath` method with the key.

   ```
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `save` method the file path.

```
    self.model.save(file_name)
```

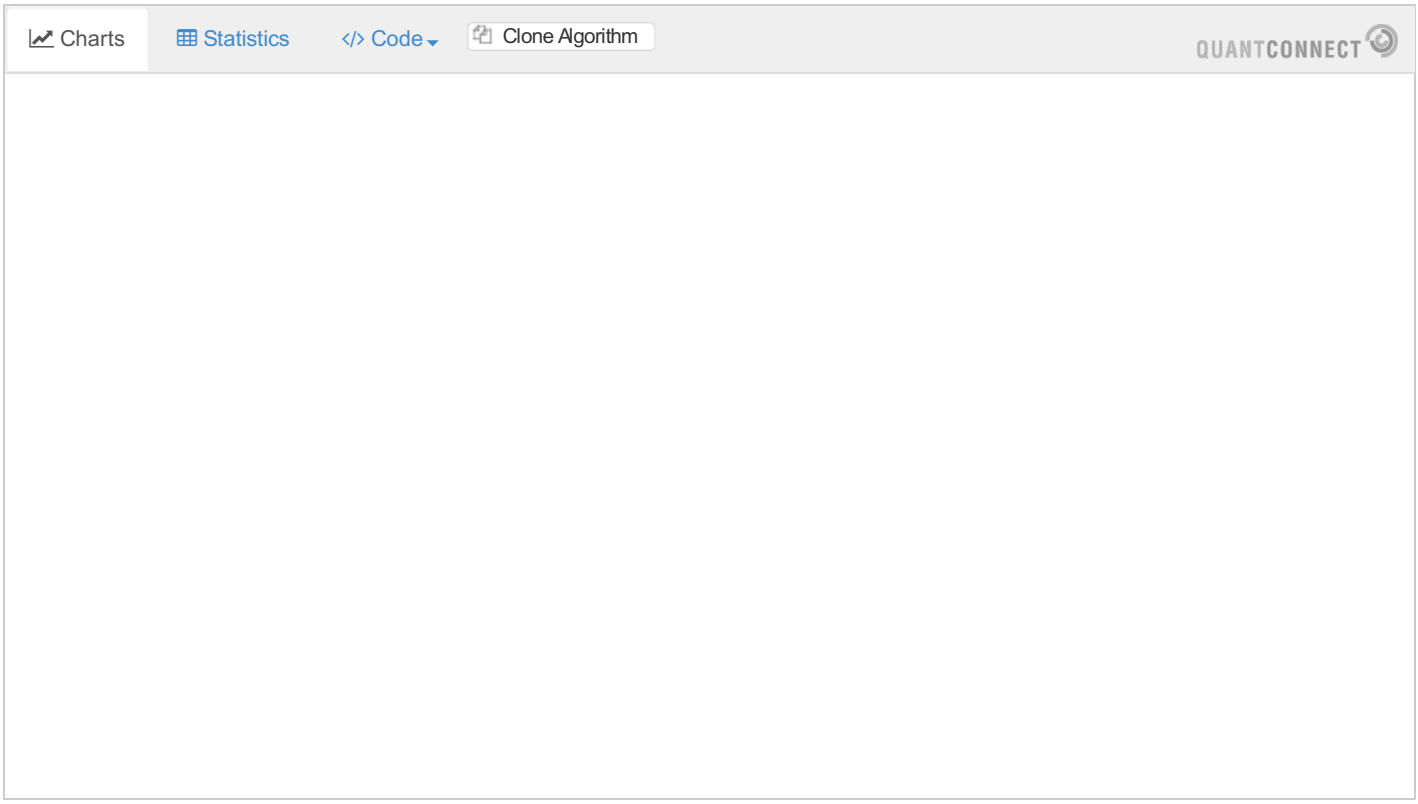
Load Models

You can load and trade with pre-trained `keras` models that you saved in the ObjectStore. To load a `keras` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load_model` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = DQN.load(file_name, env=env)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm



Popular Libraries

Tensorflow

Introduction

This page explains how to build, train, deploy and store `Tensorflow` v1 models. To view the tutorial on Tensorflow 2, see Keras .

Import Libraries

Import the `tensorflow` libraries.

```
PY
from AlgorithmImports import *
import tensorflow.compat.v1 as tf
from google.protobuf import json_format
import json5

tf.disable_v2_behavior()
```

You need the `google.protobuf` and `json5` libraries to store and load models.

Disable `tensorflow` v2 behaviors in order to deploy a v1 model.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `tensorflow` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a neural-network regression prediction model that uses the following features and labels:

Data Category	Description
Features	The last 5 closing prices
Labels	The following day's closing price

The following image shows the time difference between the features and labels:

Follow these steps to create a method to build the model:

1. Create a method to build the model for the algorithm class.

```PY
def BuildModel(self):
    # Instantiate a tensorflow session
    sess = tf.Session()

    # Declare the number of factors and then create placeholders for the input and output layers.
    num_factors = 5
    X = tf.placeholder(dtype=tf.float32, shape=[None, num_factors], name='X')
    Y = tf.placeholder(dtype=tf.float32, shape=[None])

    # Set up the weights and bias initializers for each layer.
    weight_initializer = tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform",
scale=1)
    bias_initializer = tf.zeros_initializer()

    # Create hidden layers that use the Relu activator.
    num_neurons_1 = 32
    num_neurons_2 = 16
    num_neurons_3 = 8

    W_hidden_1 = tf.Variable(weight_initializer([num_factors, num_neurons_1]))
    bias_hidden_1 = tf.Variable(bias_initializer([num_neurons_1]))
    hidden_1 = tf.nn.relu(tf.add(tf.matmul(X, W_hidden_1), bias_hidden_1))

    W_hidden_2 = tf.Variable(weight_initializer([num_neurons_1, num_neurons_2]))
    bias_hidden_2 = tf.Variable(bias_initializer([num_neurons_2]))
    hidden_2 = tf.nn.relu(tf.add(tf.matmul(hidden_1, W_hidden_2), bias_hidden_2))

    W_hidden_3 = tf.Variable(weight_initializer([num_neurons_2, num_neurons_3]))
    bias_hidden_3 = tf.Variable(bias_initializer([num_neurons_3]))
    hidden_3 = tf.nn.relu(tf.add(tf.matmul(hidden_2, W_hidden_3), bias_hidden_3))

    # Create the output layer and give it a name, so it is accessible after saving and loading the
model.
    W_out = tf.Variable(weight_initializer([num_neurons_3, 1]))
    bias_out = tf.Variable(bias_initializer([1]))
    output = tf.transpose(tf.add(tf.matmul(hidden_3, W_out), bias_out), name='outer')

    # Set up the loss function and optimizers for gradient descent optimization and backpropagation.
    # This example uses mean-square error as the loss function because the close price is a
continuous data and uses Adam as the optimizer because of its adaptive step size.
    loss = tf.reduce_mean(tf.squared_difference(output, Y))
    optimizer = tf.train.AdamOptimizer().minimize(loss)

    return sess, X, Y, output, optimizer
```

2. Instantiate the model, input layers, output layer, and optimizer and then save them as class variables.

```
PY
self.model, self.X, self.Y, self.output, self.optimizer = self.BuildModel(features, labels)
```

3. Call the `run` method with the result from the `global_variables_initializer` method.

```
PY
self.model.run(tf.global_variables_initializer())
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
PY
training_length = 252*2
self.training_data = RollingWindow[float](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar.Close)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
PY
def get_features_and_labels(self, n_steps=5):
    close_prices = list(self.training_data)[::-1]

    features = []
    labels = []
    for i in range(len(close_prices)-n_steps):
        features.append(close_prices[i:i+n_steps])
        labels.append(close_prices[i+n_steps])
    features = np.array(features)
    labels = np.array(labels)

    return features, labels

def my_training_method(self):
    features, labels = self.get_features_and_labels()
    self.model.run(self.optimizer, feed_dict={self.X: features, self.Y: labels})
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
PY
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current close price to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol].Close)
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `run` method with new features.

```
new_features, __ = self.get_features_and_labels()
prediction = self.model.run(self.output, feed_dict={self.X: new_features[-1].reshape(1, -1)})
prediction = float(prediction.flatten()[-1])
```

You can use the label prediction to place orders.

```
if prediction > slice[self.symbol].Price:
    self.SetHoldings(self.symbol, 1)
else:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `Tensorflow` models into the ObjectStore :

1. Export the `TensorFlow` graph as a JSON object.

   ```
   graph_definition = tf.compat.v1.train.export_meta_graph()
   json_graph = json_format.MessageToJson(graph_definition)
   ```

2. Export the `TensorFlow` weights as a JSON object.

   ```
   weights = self.model.run(tf.compat.v1.trainable_variables())
   weights = [w.tolist() for w in weights]
   json_weights = json5.dumps(weights)
   ```

3. Save the graph and weights to the `ObjectStore` .

```
PY

    self.ObjectStore.Save('graph', json_graph)
    self.ObjectStore.Save('weights', json_weights)
```

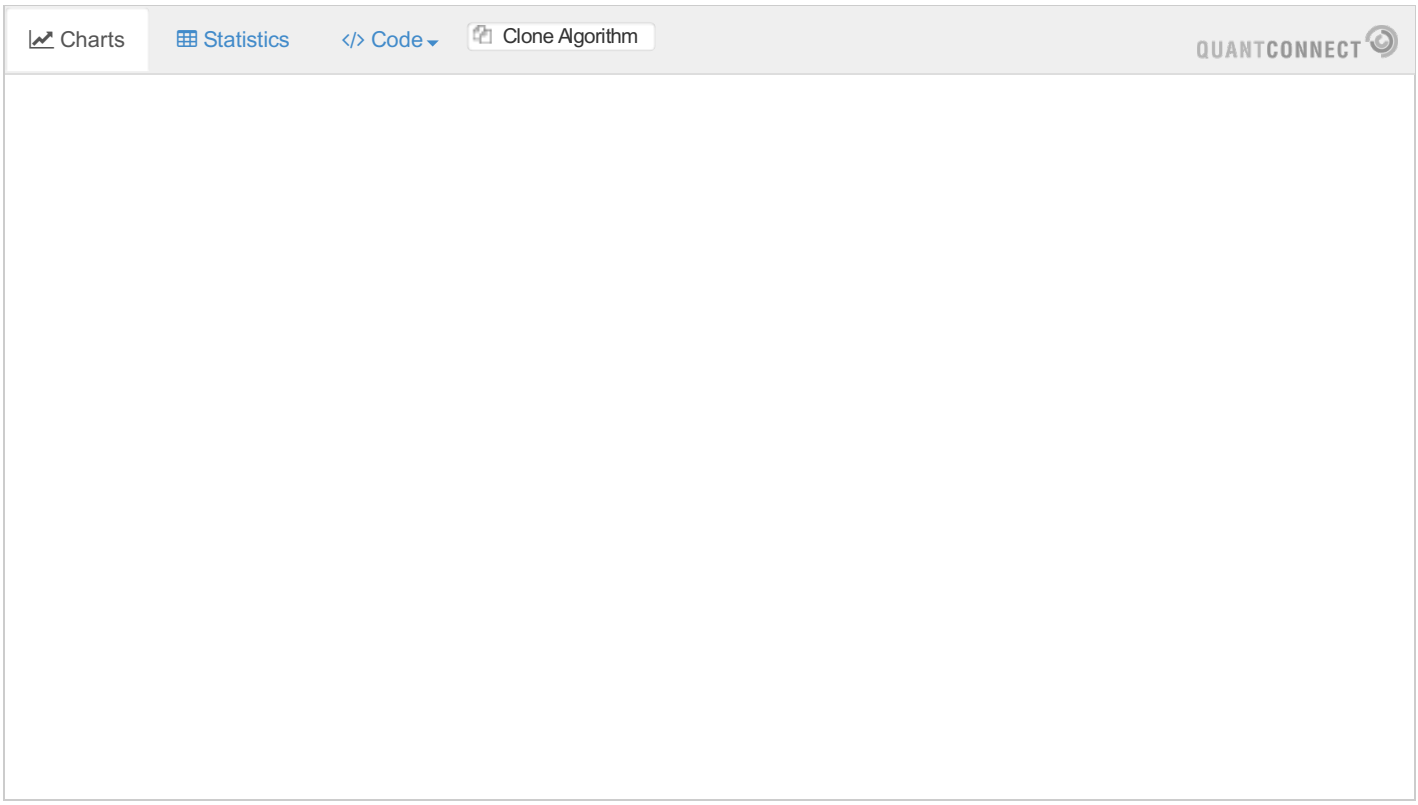
Load Models

You can load and trade with pre-trained `tensorflow` models that you saved in the ObjectStore. To load a `tensorflow` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then recall the graph and weights of the model.

```
PY

def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey('graph') and self.ObjectStore.ContainsKey('weights'):
        json_graph = self.ObjectStore.Read('graph')
        json_weights = self.ObjectStore.Read('weights')

        # Restore the tensorflow graph from JSON objects
        tf.reset_default_graph()
        graph_definition = json_format.Parse(json_graph, tf.MetaGraphDef())
        self.model = tf.Session()
        tf.train.import_meta_graph(graph_definition)

        # Select the input, output tensors and optimizer
        self.X = tf.get_default_graph().get_tensor_by_name('X:0')
        self.Y = tf.get_default_graph().get_tensor_by_name('Y:0')
        self.output = tf.get_default_graph().get_tensor_by_name('outer:0')
        self.optimizer = tf.get_default_graph().get_collection('Variable/Adam')

        # Restore the model weights from the JSON object.
        weights = [np.asarray(x) for x in json5.loads(json_weights)]
        assign_ops = []
        feed_dict = {}
        vs = tf.trainable_variables()
        zipped_values = zip(vs, weights)
        for var, value in zipped_values:
            value = np.asarray(value)
            assign_placeholder = tf.placeholder(var.dtype, shape=value.shape)
            assign_op = var.assign(assign_placeholder)
            assign_ops.append(assign_op)
            feed_dict[assign_placeholder] = value
        self.model.run(assign_ops, feed_dict=feed_dict)
```

The `ContainsKey` method returns a boolean that represents if the `graph` and `weights` is in the ObjectStore. If the ObjectStore does not contain the keys, save the model using them before you proceed.

Clone Example Algorithm


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Popular Libraries

Tslearn

Introduction

This page explains how to build, train, deploy and store `Tslearn` models.

Import Libraries

Import the `tslearn` libraries.

```
PY
from AlgorithmImports import *
from tslearn.barycenters import softdtw_barycenter
from tslearn.clustering import TimeSeriesKMeans
```

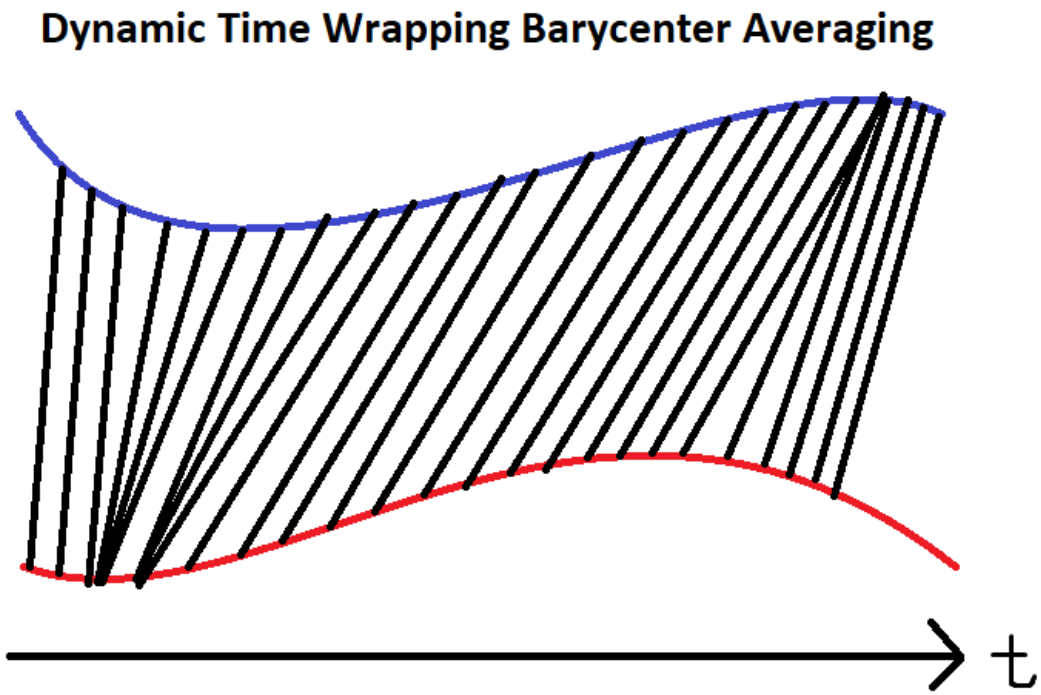
Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `tslearn` model.

```
PY
tickers = ["SPY", "QQQ", "DIA",
           "AAPL", "MSFT", "TSLA",
           "IEF", "TLT", "SHV", "SHY",
           "GLD", "IAU", "SLV",
           "USO", "XLE", "XOM"]
symbols = [self.AddEquity(ticker, Resolution.Daily).Symbol for ticker in tickers]
```

Build Models

In this example, train a model that clusters the universe of Equities into distinct groups and then allocate an equal portion of the portfolio to each cluster. To cluster the securities, instead of using a real-time comparison, apply Dynamic Time Wrapping Barycenter Averaging (DBA) to their historical prices and then run a k-means clustering algorithm. DBA is a technique of averaging a few time-series into a single one without losing much of their information. Since not all time-series move efficiently like in ideal EMH assumption, this technique allows similarity analysis of different time-series with sticky lags. The following image shows a visualization of the process. For more information about the technical details, see Dynamic Time Warping in the tslearn documentation .



To perform DBA and then cluster the securities by k-means, create a TimeSeriesKMeans model:

```
PY
self.model = TimeSeriesKMeans(n_clusters=6,   # We have 6 main groups
                    metric="dtw")
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
PY
training_length = 252
self.training_data = {}
history = self.History(self.symbols, training_length, Resolution.Daily).unstack(0).close
for symbol in self.symbols:
    self.training_data[symbol] = RollingWindow[float](training_length)
    for close_price in history[symbol]:
        self.training_data[symbol].Add(close_price)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
def get_features(self):
    close_price = pd.DataFrame({symbol: list(data)[::-1] for symbol, data in self.training_data.items()})
    log_price = np.log(close_price)
    log_normal_price = (log_price - log_price.mean()) / log_price.std()

    return log_normal_price

def my_training_method(self):
    features = self.get_features()
    self.model.fit(features.T.values)
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    for kvp in slice.Bars:
        self.training_data[kvp.Key].Add(kvp.Value.Close)
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
features = self.get_features()
self.labels = self.model.predict(features.T.values)
```

You can use the label prediction to place orders.

```
for i in set(self.labels):
    assets_in_cluster = features.columns[[n for n, k in enumerate(self.labels) if k == i]]
    size = 1/6/len(assets_in_cluster)
    self.SetHoldings([PortfolioTarget(symbol, size) for symbol in assets_in_cluster])
```

Save Models

Follow these steps to save `tslearn` models into the ObjectStore :

1. Set the key name of the model to be stored in the ObjectStore.

```
model_key = "model"
```

2. Call the `GetFilePath` method with the key.

```
file_name = self.ObjectStore.GetFilePath(model_key)
```

   This method returns the file path where the model will be stored.

3. Call the `to_hdf5` method with the file path.

```
self.model.to_hdf5(file_name + ".hdf5")
```

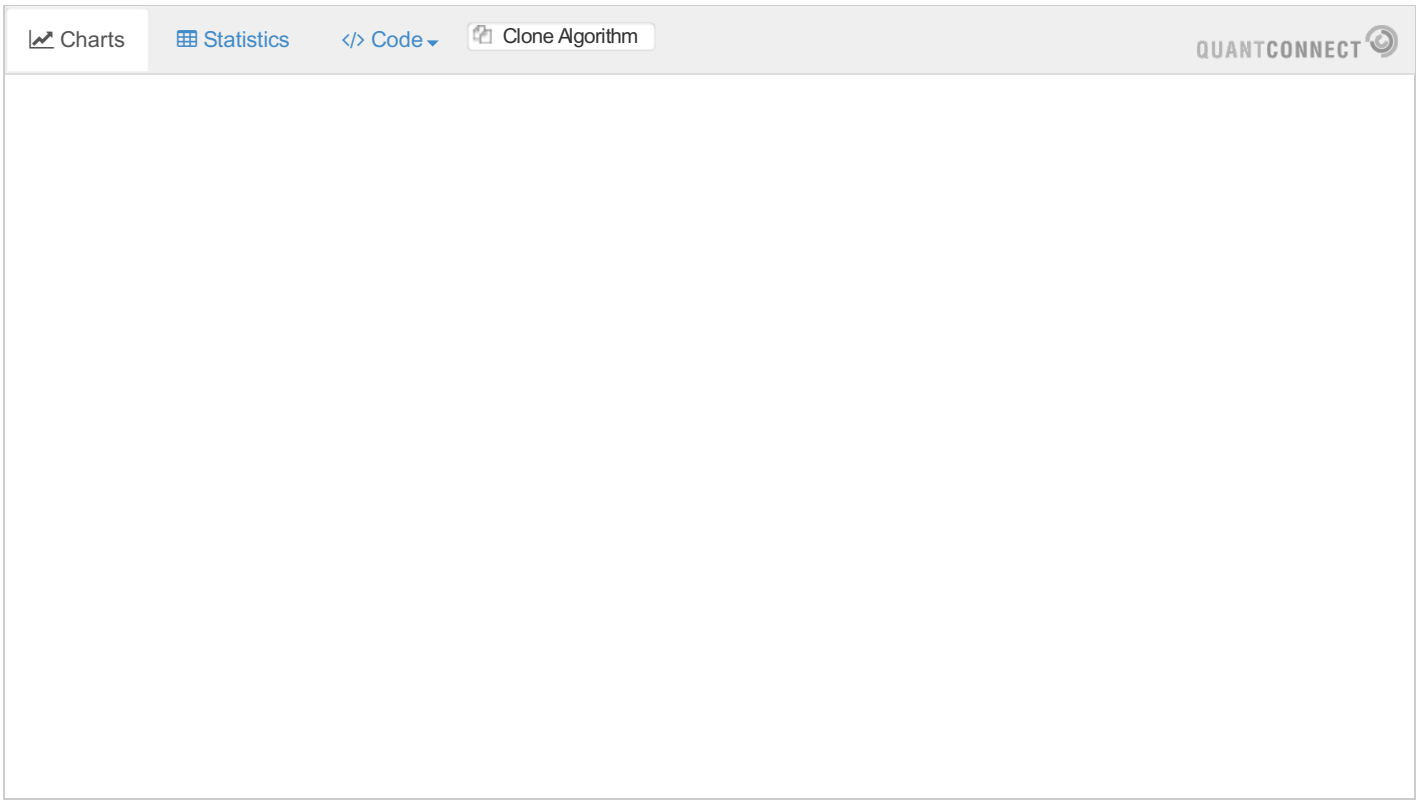
Load Models

You can load and trade with pre-trained `tslearn` models that saved in ObjectStore. To load a `tslearn` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `from_hdf5` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = TimeSeriesKMeans.from_hdf5(file_name + ".hdf5")
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm


Charts
Statistics
Code
Clone Algorithm
QUANTCONNECT

Popular Libraries

XGBoost

Introduction

This page explains how to build, train, deploy and store `XGBoost` models.

Import Libraries

Import the `XGBoost` and `joblib` libraries.

```
PY
from AlgorithmImports import *
import xgboost as xgb
import joblib
```

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `xgboost` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a gradient boost tree regression prediction model that uses the following features and labels:

Data Category	Description
Features	The last 5 closing prices
Labels	The following day's closing price

The following image shows the time difference between the features and labels:

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
training_length = 252*2
self.training_data = RollingWindow[float](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar.Close)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
PY

def get_features_and_labels(self, n_steps=5):
    close_prices = np.array(list(self.training_data)[::-1])
    df = (np.roll(close_prices, -1) - close_prices) * 0.5 + close_prices * 0.5
    df = df[:-1]

    features = []
    labels = []
    for i in range(len(df)-n_steps):
        features.append(df[i:i+n_steps])
        labels.append(df[i+n_steps])

    features = np.array(features)
    labels = np.array(labels)
    features = (features - features.mean()) / features.std()
    labels = (labels - labels.mean()) / labels.std()

    d_matrix = xgb.DMatrix(features, label=labels)

    return d_matrix

def my_training_method(self):
    d_matrix = self.get_features_and_labels()
    params = {
        'booster': 'gbtree',
        'colsample_bynode': 0.8,
        'learning_rate': 0.1,
        'lambda': 0.1,
        'max_depth': 5,
        'num_parallel_tree': 100,
        'objective': 'reg:squarederror',
        'subsample': 0.8,
      }
    self.model = xgb.train(params, d_matrix, num_boost_round=10)
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
PY

self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
PY

# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
PY

def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol].Close)
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
new_d_matrix = self.get_features_and_labels(df)
prediction = self.model.predict(new_d_matrix)
prediction = prediction.flatten()
```

You can use the label prediction to place orders.

```
if float(prediction[-1]) > float(prediction[-2]):
    self.SetHoldings(self.symbol, 1)
else:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `xgboost` models into the ObjectStore :

1. Set the key name of the model to be stored in the ObjectStore.

   ```
   model_key = "model"
   ```

2. Call the `GetFilePath` method with the key.

   ```
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   ```
   joblib.dump(self.model, file_name)
   ```

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

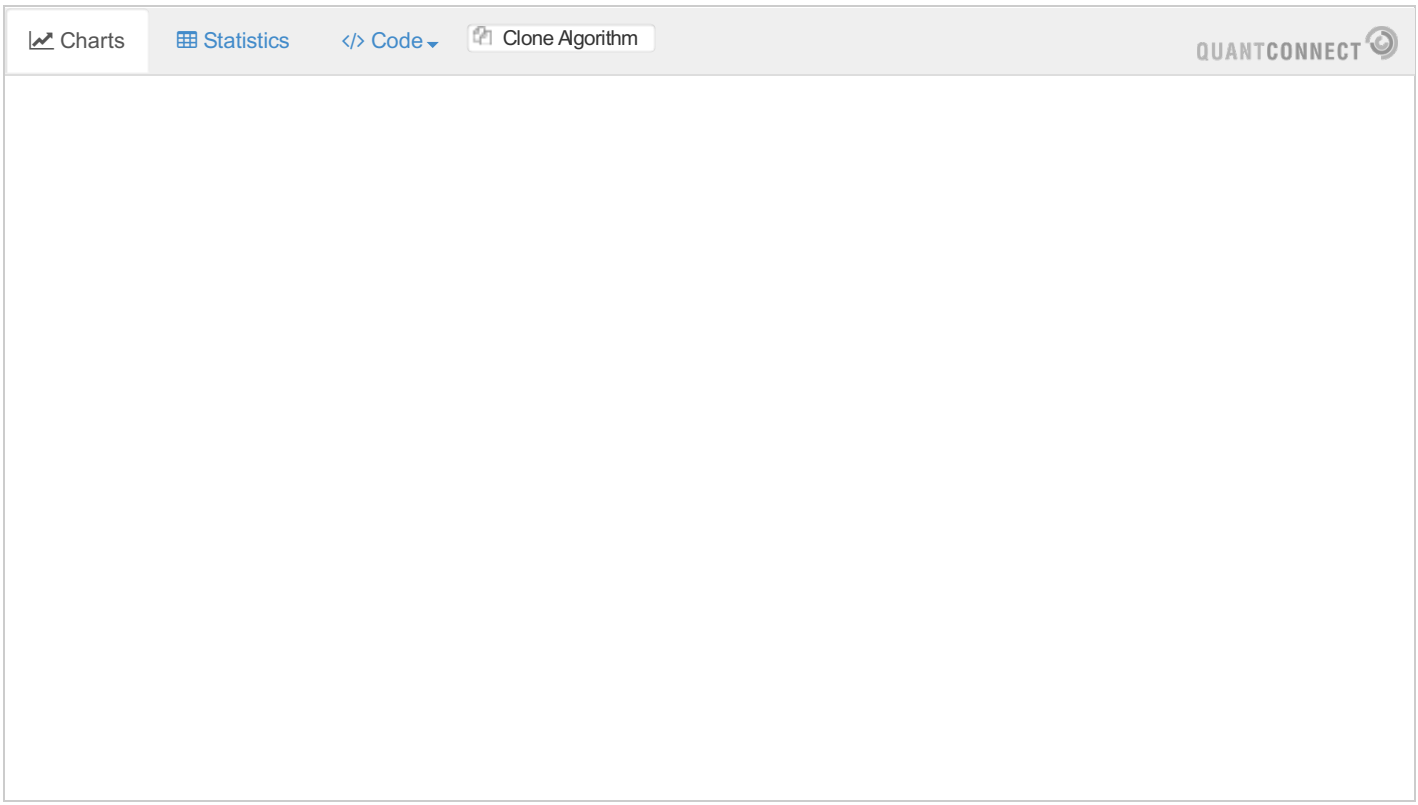
You can load and trade with pre-trained `xgboost` models that saved in ObjectStore. To load a `xgboost` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = joblib.load(file_name)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore

does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm



Popular Libraries

Aesera

Introduction

This page explains how to build, train, deploy and store `Aesera` models.

Import Libraries

Import the `aesera` and `sklearn` libraries.

```
PY
from AlgorithmImports import *
import aesara
import aesara.tensor as at
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import joblib
```

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `sklearn` model and make predictions.

```
PY
self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
```

Build Models

In this example, build a logistic regression prediction model that uses the following features and labels:

Data Category	Description
Features	Normalized daily close price of the SPY over the last 5 days
Labels	Return direction of the SPY over the next day

The following image shows the time difference between the features and labels:

Follow the below steps to build the model:

1. Initialize variables.

```
PY

# Declare Aesara symbolic variables
x = at.dmatrix("x")
y = at.dvector("y")

# initialize the weight vector w randomly using share so model coefficients keep their values
# between training iterations (updates)
rng = np.random.default_rng(100)
w = aesara.shared(rng.standard_normal(5), name="w")

# initialize the bias term
b = aesara.shared(0., name="b")
```

2. Construct the model graph.

```
PY

# Construct Aesara expression graph
p_1 = 1 / (1 + at.exp(-at.dot(x, w) - b))       # Logistic transformation
prediction = p_1 > 0.5                          # The prediction thresholded
xent = y * at.log(p_1) - (1 - y) * at.log(1 - p_1)  # Cross-entropy log-loss function
cost = xent.mean() + 0.01 * (w ** 2).sum()      # The cost to minimize (MSE)
gw, gb = at.grad(cost, [w, b])                  # Compute the gradient of the cost
```

3. Compile the model.

```
PY

self.train = aesara.function(
          inputs=[x, y],
          outputs=[prediction, xent],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))
self.predict = aesara.function(inputs=[x], outputs=prediction)
```

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

```
PY

training_length = 252*2
self.training_data = RollingWindow[TradeBar](training_length)
history = self.History[TradeBar](self.symbol, training_length, Resolution.Daily)
for trade_bar in history:
    self.training_data.Add(trade_bar)
```

Define a Training Method

To train the model, define a method that fits the model with the training data.

```
PY

def get_features_and_labels(self, n_steps=5):
    training_df = self.PandasConverter.GetDataFrame[TradeBar](list(self.training_data)[::-1])['close']

    features = []
    for i in range(1, n_steps + 1):
        close = training_df.shift(i)[n_steps:-1]
        close.name = f"close-{i}"
        features.append(close)
    features = pd.concat(features, axis=1)
    # Normalize using the 5 day interval
    features = MinMaxScaler().fit_transform(features.T).T[4:]

    Y = training_df.pct_change().shift(-1)[n_steps*2-1:-1].reset_index(drop=True)
    labels = np.array([1 if y > 0 else 0 for y in Y])   # binary class

    return features, labels

def my_training_method(self):
    features, labels = self.get_features_and_labels()
    D = (features, labels)
    self.train(D[0], D[1])
```

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

```
PY

self.Train(self.my_training_method)
```

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

```
PY

# Train the model every Sunday at 8:00 AM
self.Train(self.DateRules.Every(DayOfWeek.Sunday), self.TimeRules.At(8, 0), self.my_training_method)
```

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

```
def OnData(self, slice: Slice) -> None:
    if self.symbol in slice.Bars:
        self.training_data.Add(slice.Bars[self.symbol])
```

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

```
features, _ = self.get_features_and_labels()
prediction = self.predict(features[-1].reshape(1, -1))
prediction = float(prediction)
```

You can use the label prediction to place orders.

```
if prediction == 1:
    self.SetHoldings(self.symbol, 1)
elif prediction == 0:
    self.SetHoldings(self.symbol, -1)
```

Save Models

Follow these steps to save `sklearn` models into the ObjectStore :

1. Set the key name you want to store the model under in the ObjectStore.

   ```
   model_key = "model"
   ```

2. Call the `GetFilePath` method with the key.

   ```
   file_name = self.ObjectStore.GetFilePath(model_key)
   ```

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   ```
   joblib.dump(self.predict, file_name)
   ```

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

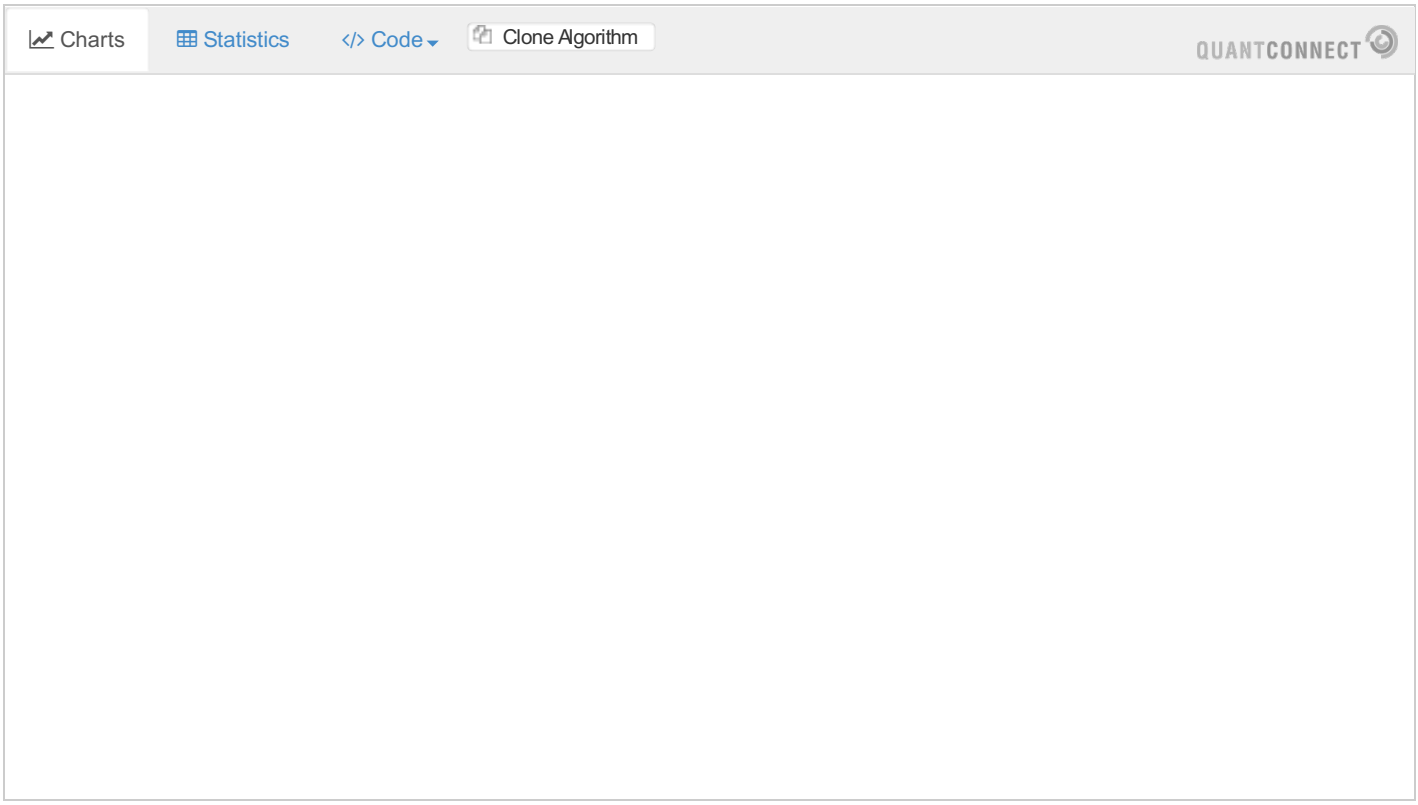
Load Models

You can load and trade with pre-trained `sklearn` models that you saved in the ObjectStore. To load a `sklearn` model from the ObjectStore, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

```
def Initialize(self) -> None:
    if self.ObjectStore.ContainsKey(model_key):
        file_name = self.ObjectStore.GetFilePath(model_key)
        self.model = joblib.load(file_name)
```

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the ObjectStore. If the ObjectStore does not contain the `model_key` , save the model using the `model_key` before you proceed.

Clone Example Algorithm



Algorithm Framework

Algorithm Framework

Overview

Introduction

Since QuantConnect was formed in 2013, the community has created more than 2 million algorithms. Although they all seek to achieve different results, they can all be abstracted into the same core features. The Algorithm Framework is our attempt to provide this well-defined structure to the community, encouraging good design and making your work more reusable. LEAN provides many pre-built Algorithm Framework models you can use in your algorithms.

Strategy Styles

The algorithm framework works best for portfolio rebalancing, ranking, and multi-factor style strategies. The default portfolio construction models are designed to weigh and invest in the assets as the signals values shift, aiming to be fully invested as much as possible.

Some strategies might find it difficult to fit this model. Technical entry signals are often tightly coupled with their exit criteria, they may not have defined hold periods, and they might follow the market to determine when to exit. This style of strategy can be addressed in a few ways:

1. Short Term Signals - Emitting insights for short periods until they are no longer valid.
2. Cancel Signals - Updating an insight's expiry time so that it's expired upon the exit signal.
3. Alpha Exit Signal - Treating exit as its own signal, a separate Alpha model could indicate exit.
4. Risk Model - Tracking exit using a risk model that attempts to lock in gains.

These techniques are still not ideal for many strategies. For example, with only a single signal, the built in `EqualWeightingPortfolioConstructionModel` will allocate 100% of buying power to a single asset, over concentrating risk. If a second signal comes a few days later, it would sell 50% of the portfolio to free cash and invest in the second asset. This can create high trading costs through portfolio churn.

With a custom portfolio construction model , signals can mean what you need for your specific strategy style. For example, you might create a " `MultiBetPortfolioConstructionModel` " that uses each insight to open a specific "bet" that is separately allocated capital and tracked.

Important Terminology

The following table defines important Algorithm Framework terminology:

Term	Description
Universe Selection model	A framework module that selects assets for your algorithm.
Alpha model	A framework module that generates trading signals on the assets in your universe.
Insight	An object that represents a trading signal. The Alpha model generates these objects for the Portfolio Construction model.
Portfolio Construction model	A framework module that determines position size targets based on the `Insight` objects it receives from the Alpha model.
PortfolioTarget	An object that represents the target position size for an asset in the universe. The Portfolio Construction model generates these objects for the Risk Management model.
Risk Management	A framework module that manages market risks and ensures the algorithm remains within target parameters. This model adjusts the `PortfolioTarget` objects it receives from the Portfolio Construction model before they reach the Execution model.
Execution	A framework module that places trades to reach the target risk-adjust portfolio based on the `PortfolioTarget` objects it receives from the Risk Management model.

Framework Advantages

We recommend you design strategies with the Algorithm Framework in most cases. The Algorithm Framework has many advantages over the classic algorithm design.

Pluggable Algorithm Modules

With the Algorithm Framework, your code can instantly utilize all modules built within the framework. The modules clip together in well-defined ways, which enable them to be shared and swapped interchangeably.

Focus On Your Strengths

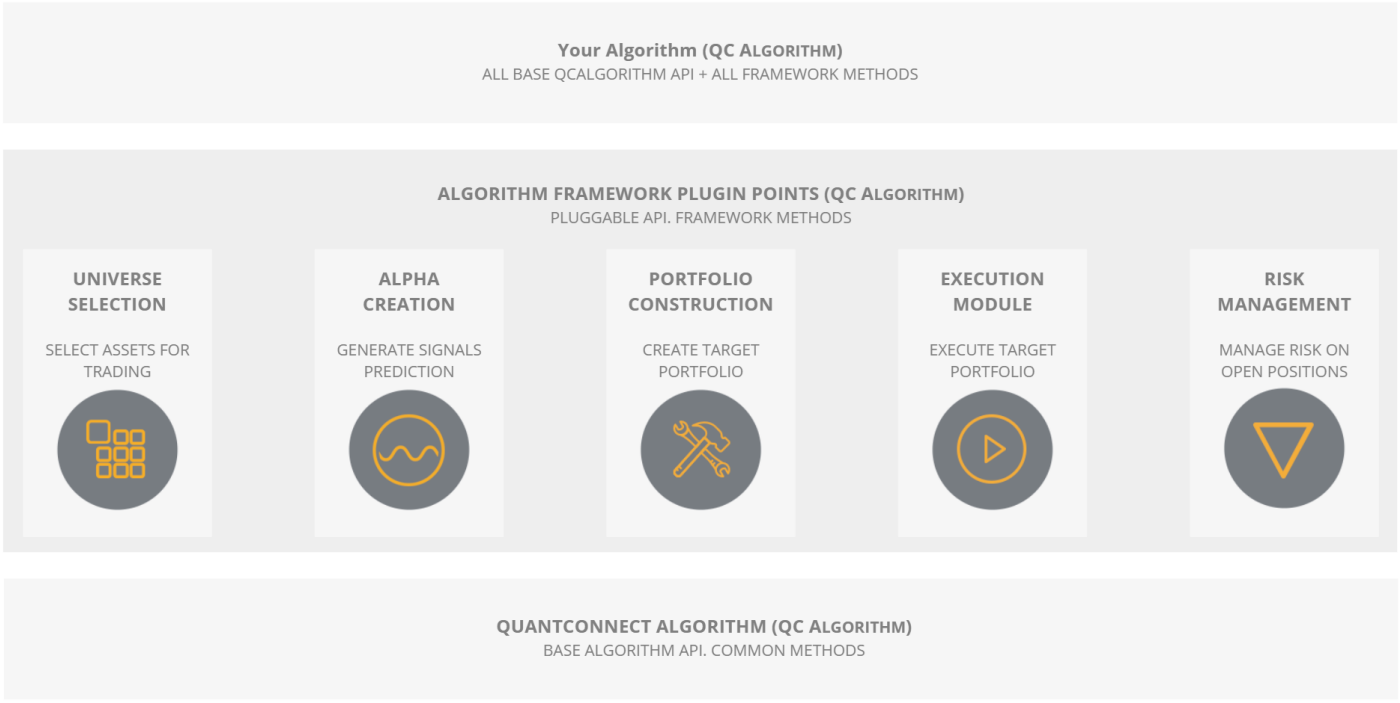
If you write code in modules, you can focus on your strengths. If you are great at building universes, you can build Universe Selection modules. If you have risk management experience, you could write reusable risk monitoring technology.

Reduce Development By Using Community Modules

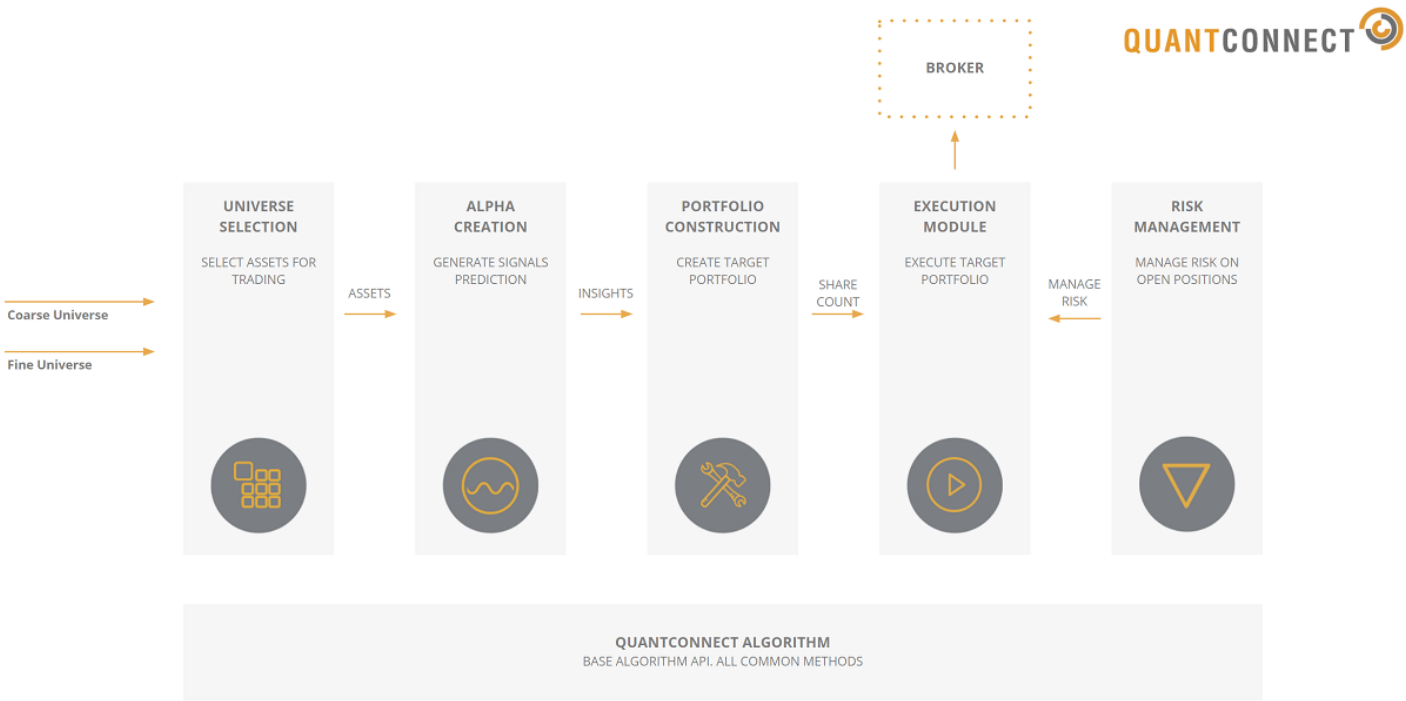
Easily share modules you've made between algorithms or pull in ones produced by the community. The strict separation of duties makes the Algorithm Framework perfect for reusing code between strategies.

System Architecture

The Algorithm Framework is built into the `QCAlgorithm` class, so your strategy can access all the normal methods you use for algorithm development. You should be able to copy and paste your algorithms across without any changes. You don't need a separate class.



The framework data output of each module flows into the following module. The assets that the Universe Selection model selects are fed into the Alpha model to generate trade signals (`Insight` objects). The `Insight` objects from the Alpha model are fed into the Portfolio Construction model to create `PortfolioTarget` objects, which contain the target number of units to hold for each asset. The `PortfolioTarget` objects from the Portfolio Construction model are fed into the Risk Management model to ensure the targets are within safe risk parameters and to adjust the `PortfolioTarget` objects if necessary. The `PortfolioTarget` objects from the Risk Management model are fed into the Execution model, which efficiently places trades to acquire the target portfolio.



```
class MyFrameworkAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetUniverseSelection()
        self.AddAlpha()
        self.SetPortfolioConstruction()
        self.SetExecution()
        self.AddRiskManagement()
```

PY

For simple strategies, it may seem like overkill to abstract out your algorithm concepts. However, even simple strategies can benefit from reusing the ecosystem of modules available in QuantConnect. Imagine pairing your EMA-cross Alpha model with a better execution system or plugging in an open-source trailing stop Risk Management model .

Separation of Concerns

The separation of concerns principle is an algorithm design principle where individual components are isolated and their responsibilities don't overlap. The Algorithm Framework is designed to uphold this design principle. The individual framework components shouldn't rely on the state of the other framework components in order to operate.

In the Algorithm Framework, the models should not communicate. The universe constituents and the `Insight` objects you emit should be the same, regardless of what you select for the Portfolio Construction, Risk Management, and Execution models. If your algorithm logic naturally violates the separation of concerns design principle, it's more appropriate to use the classic or hybrid design .

Framework vs Classic Design

Select the classic or Algorithm Framework design based on the needs of your algorithm. If your algorithm uses special order types or passes information between the different algorithm components, use the classic design. If you can reuse existing modules and the Algorithm Framework modules can work in isolation, use the Algorithm Framework design.

Algorithm Framework

Universe Selection

Universe Selection

Key Concepts

Introduction

The Universe Selection model creates `Universe` objects, which select the assets for your algorithm. As the universe changes, we notify your algorithm through the `OnSecuritiesChanged` event handler. With this event handler, you can track the current universe constituents in other parts of your algorithm without breaking the separation of concerns design principle.

Types of Universe Selection

We have identified several types of universes that cover most people's requirements and built helper classes to make their implementation easier. The following table describes the types of pre-built Universe Selection models:

Universe Type	Description
Manual Universes	Universes that use a fixed, static set of assets
Fundamental Universes	Universes for US Equities that are based on coarse price or fundamental data
Scheduled Universes	Universes that trigger on regular, custom intervals
Futures Universes	Universes that subscribe to Future chains
Option Universes	Universes that subscribe to Option chains

Add Models

To add a Universe Selection model, in the `Initialize` method, call the `AddUniverseSelection` method.

```
symbols = [Symbol.Create("SPY", SecurityType.Equity, Market.USA)]
self.AddUniverseSelection(ManualUniverseSelectionModel(symbols))
```

PY

To view all the pre-built Universe Selection models, see Types of Universe Selection .

Multi-Universe Algorithms

You can add multiple Universe Selection models to a single algorithm.

```
self.AddUniverseSelection(EmaCrossUniverseSelectionModel())
self.AddUniverseSelection(FineFundamentalUniverseSelectionModel(self.SelectCoarse, self.SelectFine))
```

If you add multiple Universe Selection models, the algorithm subscribes to the constituents of all the models.

Model Structure

Universe Selection models should extend the `UniverseSelectionModel` class. Extensions of the `UniverseSelectionModel` class must implement the `CreateUniverses` method, which receives an `algorithm` object and returns an array of `Universe` objects.

```
class MyUniverseSelectionModel(UniverseSelectionModel):

    # Creates the universes for this algorithm
    def CreateUniverses(self, algorithm: QCAlgorithm) -> List[Universe]:
        universes = []
        return universes

    # Gets the next time the framework should invoke the `CreateUniverses` method to refresh the set of
universes.
    def GetNextRefreshTimeUtc(self):
        return datetime.max
```

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

Generally, you should be able to extend one of the pre-built Universe Selection types. If you need to do something that doesn't fit into the pre-built types, let us know and we'll create a new foundational type of Universe Selection model.

Track Security Changes

When the Universe Selection model adjusts the universe constituents, we notify your algorithm through the `OnSecuritiesChanged` event handler on your other framework components. The method receives `QCAlgorithm` and `SecurityChanges` objects. The `QCAlgorithm` object is an instance of the base QCAlgorithm class, not a reference to your algorithm object. To access the added securities, check the `changes.AddedSecurities` property. To access the removed securities, check the `changes.RemovedSecurities` property.

```
def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
    for security in changes.AddedSecurities:
        self.Log(f"Added {security.Symbol}")

    for security in changes.RemovedSecurities:
        self.Log(f"Removed {security.Symbol}")
```

Live Trading Considerations

In live trading, the securities in your brokerage account are added to your user-defined universe. If your Universe Selection model doesn't select these securities, they are not removed from your user-defined universe. When you liquidate the positions, you can remove them from the user-defined universe. If you call the `RemoveSecurity` method, it automatically liquidates the position.

To see the securities in your user-defined universe that were loaded from your brokerage account, check your algorithm's ActiveSecurities in the OnWarmupFinished event handler.

Examples

Demonstration Algorithms
Manual Selection - BasicTemplateFrameworkAlgorithm.py Python Fundamental Selection - QC500UniverseSelectionModel.py Python Scheduled Selection - ScheduledUniverseSelectionModelRegressionAlgorithm.py Python

Universe Selection

Universe Settings

Introduction

Universe settings and security initializers enable you to configure some properties of the securities in a universe.

Properties

The universe settings of your algorithm configure some properties of the universe constituents. The following table describes the properties of the `UniverseSettings` object:

Property: `ExtendedMarketHours` Should assets also feed extended market hours? You only receive extended market hours data if you create the subscription with an intraday resolution. If you create the subscription with daily resolution, the daily bars only reflect the regular trading hours. Data Type: `bool` | Default Value: `False`
Property: `FillForward` Should asset data fill forward? Data Type: `bool` | Default Value: `True`
Property: `MinimumTimeInUniverse` What's the minimum time assets should be in the universe? Data Type: `timedelta` | Default Value: `timedelta(1)`
Property: `Resolution` What resolution should assets use? Data Type: `Resolution` | Default Value: `Resolution.Minute`

Property: `ContractDepthOffset`

What offset from the current front month should be used for continuous Future contracts ? 0 uses the front month and 1 uses the back month contract. This setting is only available for Future assets.

Data Type: `int` | Default Value: `0`

Property: `DataMappingMode`

How should continuous Future contracts be mapped? This setting is only available for Future assets.

Data Type: `DataMappingMode` | Default Value: `DataMappingMode.OpenInterest`

Property: `DataNormalizationMode`

How should historical prices be adjusted? This setting is only available for Equity and Futures assets.

Data Type: `DataNormalizationMode` | Default Value: `DataNormalizationMode.Adjusted`

Property: `Leverage`

What leverage should assets use in the universe? This setting is not available for derivative assets.

Data Type: `float` | Default Value: `Security.NullLeverage`

To set the `UniverseSettings` , update the preceding properties in the `Initialize` method before you add the "Universe Selection model" These settings are globals, so they apply to all universes you create.

```py
# Request second resolution data. This will be slow!
self.UniverseSettings.Resolution = Resolution.Second
self.AddUniverseSelection(VolatilityETFUniverse())
```

Configure Securities

Instead of configuring global universe settings, you can individually configure the settings of each security in the universe with a security initializer. Security initializers let you apply any security-level reality model or special data requests on a per-security basis. To set the security initializer, in the `Initialize` method, call the `SetSecurityInitializer` method and then define the security initializer.

```py
#In Initialize
self.SetSecurityInitializer(self.CustomSecurityInitializer)

def CustomSecurityInitializer(self, security: Security) -> None:
    # Disable trading fees
    security.SetFeeModel(ConstantFeeModel(0, "USD"))
```

For simple requests, you can use the functional implementation of the security initializer. This style lets you configure the security object with one line of code.

```py
self.SetSecurityInitializer(lambda security: security.SetFeeModel(ConstantFeeModel(0, "USD")))
```

In some cases, you may want to trade a security in the same time loop that you create the security subscription. To avoid errors, use a security initializer to set the market price of each security to the last known price. The `GetLastKnownPrices` method seeds the security price by gathering the security data over the last 3 days. If there is no data during this period, the security price remains at 0.

```py
seeder = FuncSecuritySeeder(self.GetLastKnownPrices)
self.SetSecurityInitializer(lambda security: seeder.SeedSecurity(security))
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage
- Fee
- Buying Power
- Settlement
- Margin Interest Rate

The default security initializer also sets the leverage of each security and intializes each security with a seeder function. To extend upon the default security initializer instead of overwriting it, create a custom `BrokerageModelSecurityInitializer` .

```py
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite some of the reality models
        security.SetFeeModel(ConstantFeeModel(0, "USD"))
```

To set a seeder function without overwriting the reality models of the brokerage, use the standard `BrokerageModelSecurityInitializer`.

```py
self.SetSecurityInitializer(BrokerageModelSecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))
```

Universe Selection

Manual Universes

Introduction

The `ManualUniverseSelectionModel` selects a static, fixed set of assets. It is similar to adding securities with the traditional `AddSecurity` API methods. If your algorithm has a static universe, you can use automatic indicators instead of manual indicators in your algorithm.

Manual universes can be prone to look-ahead bias . For example, if you select a set of securities that have performed well during the backtest period, you are incorporating information from the future into the backtest and the algorithm may underperform in live mode.

Add Manual Universe Selection

To add a `ManualUniverseSelectionModel` to your algorithm, in the `Initialize` method, call the `AddUniverseSelection` method. The `ManualUniverseSelectionModel` constructor expects a list of `Symbol` objects that represent the universe constituents. To create the `Symbol` objects, call the `Symbol.Create` method.

```
PY
tickers = ["SPY", "QQQ", "IWM"]
symbols = [ Symbol.Create(ticker, SecurityType.Equity, Market.USA) for ticker in tickers]
self.AddUniverseSelection(ManualUniverseSelectionModel(symbols))
```

To move the universe tickers and `Symbol` objects outside of the algorithm class, create a universe selection model that inherits the `ManualUniverseSelectionModel` class.

```
PY
# In Initialize
self.AddUniverseSelection(IndexUniverseSelectionModel())

# Outside of the algorithm class
class IndexUniverseSelectionModel(ManualUniverseSelectionModel):
    def __init__(self):
        tickers = ["SPY", "QQQ", "IWM"]
        symbols = [Symbol.Create(ticker, SecurityType.Equity, Market.USA) for ticker in tickers]
        super().__init__(symbols)
```

To view the implementation of this model, see the LEAN GitHub repository .

Universe Selection

Fundamental Universes

Introduction

A `FundamentalUniverseSelectionModel` selects a universe of US Equities based on `CoarseFundamental` and, sometimes, `FineFundamental` data. If the model uses `CoarseFundamental` data, it relies on the US Equity Coarse Universe dataset. If the Universe Selection model uses `FineFundamental` data, it relies on the US Fundamental dataset.

These types of universes operate on daily schedule. In backtests, they select assets at midnight. In live trading, the selection timing depends on the data feed you use.

If you use a fundamental Universe Selection model, the only way to unsubscribe from a security is to return a list from the selection functions that doesn't include the security `Symbol` . The `RemoveSecurity` method doesn't work with these types of Universe Selection models.

Coarse Fundamental Selection

The `CoarseFundamentalUniverseSelectionModel` selects assets based on `CoarseFundamental` data. To use this model, define a coarse selection function. The coarse selection function receives a list of `CoarseFundamental` objects and returns a list of `Symbol` objects. The `Symbol` objects you return from the coarse selection function are the constituents of the universe.

```
PY
def Initialize(self) -> None:
    self.AddUniverseSelection(CoarseFundamentalUniverseSelectionModel(self.SelectCoarse))

def SelectCoarse(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    selected = [c for c in coarse if c.HasFundamentalData]
    sorted_by_dollar_volume = sorted(selected, key=lambda c: c.DollarVolume, reverse=True)
    return [c.Symbol for c in sorted_by_dollar_volume[:100]] # Return most liquid assets w/ fundamentals
```

To move the coarse selection function outside of the algorithm class, create a universe selection model that inherits the `CoarseFundamentalUniverseSelectionModel` class.

```
PY
# In Initialize
self.AddUniverseSelection(MostLiquidFundamentalUniverseSelectionModel(self.UniverseSettings))

# Outside of the algorithm class
class MostLiquidFundamentalUniverseSelectionModel(CoarseFundamentalUniverseSelectionModel):
    def __init__(self, universe_settings: UniverseSettings) -> None:
        super().__init__(self.SelectCoarse, universe_settings)

    def SelectCoarse(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        selected = [c for c in coarse if c.HasFundamentalData]
        sorted_by_dollar_volume = sorted(selected, key=lambda c: c.DollarVolume, reverse=True)
        return [c.Symbol for c in sorted_by_dollar_volume[:100]]
```

To return the current universe constituents from the coarse selection function, return `Universe.Unchanged` .

To view the implementation of this model, see the LEAN GitHub repository .

Fine Fundamental Selection

The `FineFundamentalUniverseSelectionModel` selects assets based on `CoarseFundamental` and `FineFundamental` data. This is the only model that provides corporate fundamental data to your algorithm. To use this model, define a coarse selection function and a fine selection function. The coarse selection function receives a list of `CoarseFundamental` objects and returns a list of `Symbol` objects. To filter the `CoarseFundamental` down to the securities that have fundamental data, add a `HasFundamentalData` filter to the coarse selection function. The fine selection function receives a subset of `FineFundamental` objects generated from coarse selection results and returns a list of `Symbol` objects. The `Symbol` objects you return from the fine selection function are the constituents of the universe.

```
PY

def Initialize(self) -> None:
    self.AddUniverseSelection(FineFundamentalUniverseSelectionModel(self.SelectCoarse, self.SelectFine))

def SelectCoarse(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
    selected = [c for c in coarse if c.HasFundamentalData]
    sorted_by_dollar_volume = sorted(selected, key=lambda c: c.DollarVolume, reverse=True)
    return [c.Symbol for c in sorted_by_dollar_volume[:100]] # Return most liquid assets w/ fundamentals

def SelectFine(self, fine: List[FineFundamental]) -> List[Symbol]:
    sorted_by_pe_ratio = sorted(fine, key=lambda x: x.ValuationRatios.PERatio, reverse=False)
    return [c.Symbol for c in sorted_by_pe_ratio[:10]] # Return assets with lowest P/E ratios
```

To move the selection functions outside of the algorithm class, create a universe selection model that inherits the `FineFundamentalUniverseSelectionModel` class.

```
PY

# In Initialize
self.AddUniverseSelection(LowPERatioUniverseSelectionModel(self.UniverseSettings))

# Outside of the algorithm class
class LowPERatioUniverseSelectionModel(FineFundamentalUniverseSelectionModel):
    def __init__(self, universe_settings: UniverseSettings = None) -> None:
        super().__init__(self.SelectCoarse, self.SelectFine, universe_settings)

    def SelectCoarse(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        selected = [c for c in coarse if c.HasFundamentalData]
        sorted_by_dollar_volume = sorted(selected, key=lambda c: c.DollarVolume, reverse=True)
        return [c.Symbol for c in sorted_by_dollar_volume[:100]]

    def SelectFine(self, fine: List[FineFundamental]) -> List[Symbol]:
        sorted_by_pe_ratio = sorted(fine, key=lambda x: x.ValuationRatios.PERatio, reverse=False)
        return [c.Symbol for c in sorted_by_pe_ratio[:10]]
```

To return the current universe constituents from the coarse or fine selection function, return `Universe.Unchanged` .

If you add a `FineFundamentalUniverseSelectionModel` to your algorithm, you can access fundamental data in the fine selection function or from the `Equity` object. To access fundamental data from the `Equity` object, use the `Equity.Fundamentals` property.

To view the implementation of this model, see the LEAN GitHub repository .

EMA Cross Selection

The `EmaCrossUniverseSelectionModel` applies two exponential moving average (EMA) indicators to the price history

of assets and then selects the assets that have their fast EMA furthest above their slow EMA on a percentage basis.

```
def Initialize(self) -> None:
    self.AddUniverseSelection(EmaCrossUniverseSelectionModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fastPeriod`	`int`	Fast EMA period	100
`slowPeriod`	`int`	Slow EMA period	300
`universeCount`	`int`	Maximum number of members of this universe selection	500
`universeSettings`	`UniverseSettings`	The settings used when adding symbols to the algorithm	`None`

If you don't provide a `universeSettings` argument, the `algorithm.UniverseSettings` are used by default.

To view the implementation of this model, see the LEAN GitHub repository .

Universe Selection

ETF Constituents Universes

Introduction

The `ETFConstituentsUniverseSelectionModel` selects a daily universe of US Equities based on the constituents of an ETF. These Universe Selection models rely on the US ETF Constituents dataset.

Add ETF Constituents Universe Selection

To add an `ETFConstituentsUniverseSelectionModel` to your algorithm, in the `Initialize` method, call the `AddUniverseSelection` method. The `ETFConstituentsUniverseSelectionModel` constructor expects an ETF `Symbol` .

```
self.AddUniverseSelection(ETFConstituentsUniverseSelectionModel(etfSymbol))
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`etfSymbol`	`Symbol`	`Symbol` of the ETF to get constituents for	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`None`
`universeFilterFunc`	`Callable[[List[ETFConstituentData]], List[Symbol]]`	Function to filter ETF constituents. If you don't provide an argument, the model selects all of the ETF constituents by default.	`None`

If you provide a `universeFilterFunc` argument, you can use the following attributes of the `ETFConstituentData` objects to select your universe:

The following example shows how to select the 10 Equities with the largest weight in the SPY ETF:

```
def Initialize(self) -> None:
    symbol = Symbol.Create("SPY", SecurityType.Equity, Market.USA)
    universe = ETFConstituentsUniverseSelectionModel(symbol, self.UniverseSettings,
self.ETFConstituentsFilter)
    self.AddUniverseSelection(universe)

def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
    selected = sorted([c for c in constituents if c.Weight],
        key=lambda c: c.Weight, reverse=True)[:10]
    return [c.Symbol for c in selected]
```

To move the ETF `Symbol` and the selection function outside of the algorithm class, create a universe selection model that inherits the `ETFConstituentsUniverseSelectionModel` class.

```
# In Initialize
self.AddUniverseSelection(LargestWeightSPYETFUniverseSelectionModel(self.UniverseSettings))

# Outside of the algorithm class
class LargestWeightSPYETFUniverseSelectionModel(ETFConstituentsUniverseSelectionModel):
    def __init__(self, universe_settings: UniverseSettings = None) -> None:
        symbol = Symbol.Create("SPY", SecurityType.Equity, Market.USA)
        super().__init__(symbol, universe_settings, self.ETFConstituentsFilter)

    def ETFConstituentsFilter(self, constituents: List[ETFConstituentData]) -> List[Symbol]:
        selected = sorted([c for c in constituents if c.Weight],
            key=lambda c: c.Weight, reverse=True)[:10]
        return [c.Symbol for c in selected]
```

To return the current universe constituents from the selection function, return `Universe.Unchanged` .

To view the implementation of this model, see the LEAN GitHub repository .

Universe Selection

Scheduled Universes

Introduction

A Scheduled Universe Selection model selects assets at fixed, regular intervals. In live trading, you can use this type of model to fetch tickers from Dropbox or to perform periodic historical analysis and select some assets.

Scheduled Universe Selection

The `ScheduledUniverseSelectionModel` selects assets on the schedule you provide. To use this model, provide a `DateRule` , `TimeRule` , and a selector function. The `DateRule` and `TimeRule` define the selection schedule. The selector function receives a `datetime` object and returns a list of `Symbol` objects. The `Symbol` objects you return from the selector function are the constituents of the universe.

```
self.AddUniverseSelection(ScheduledUniverseSelectionModel(dateRule, timeRule, selector))
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`dateRule`	`IDateRule`	Date rule defines what days the universe selection function will be invoked	
`timeRule`	`ITimeRule`	Time rule defines what times on each day selected by date rule the universe selection function will be invoked	
`selector`	`Callable[[datetime], List[Symbol]]`	Selector function accepting the date time firing time and returning the universe selected symbols	
`settings`	`UniverseSettings`	Universe settings for subscriptions added via this universe, null will default to algorithm's universe settings	`None`

If you don't provide a `settings` argument, the `algorithm.UniverseSettings` is used by default.

The model assumes the `timeRule` argument is in Coordinated Universal Time (UTC). To use a different timezone, pass a `timeZone` argument of type `DateTimeZone` before the `dateRule` argument.

To return the current universe constituents from the selector function, return `Universe.Unchanged` .

```py
# Selection will run on mon/tues/thurs at 00:00/06:00/12:00/18:00
self.AddUniverseSelection(ScheduledUniverseSelectionModel(
    self.DateRules.Every(DayOfWeek.Monday, DayOfWeek.Tuesday, DayOfWeek.Thursday),
    self.TimeRules.Every(timedelta(hours = 6)),
    self.SelectSymbols # selection function in algorithm.
))

# Create selection function which returns symbol objects.
def SelectSymbols(self, dateTime: datetime) -> List[Symbol]:
    tickers = ['SPY', 'AAPL', 'IBM']
    return [Symbol.Create(ticker, SecurityType.Equity, Market.USA)
        for ticker in tickers]
```

To view the implementation of this model, see the LEAN GitHub repository .

Date Rules

The following table describes the supported `DateRules` :

Member	Description
`self.DateRules.SetDefaultTimeZone(time_zone: DateTimeZone)`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`self.DateRules.On(year: int, month: int, day: int)`	Trigger an event on a specific date.
`self. DateRules.EveryDay()`	Trigger an event every day.
`self.DateRules.EveryDay(symbol: Symbol)`	Trigger an event every day a specific symbol is trading.
`self.DateRules.Every(days: List[DayOfWeek])`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`self.DateRules.MonthStart(daysOffset: int = 0)`	Trigger an event on the first day of each month plus an offset.
`self.DateRules.MonthStart(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`self.DateRules.MonthEnd(daysOffset: int = 0)`	Trigger an event on the last day of each month minus an offset.
`self.DateRules.MonthEnd(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`self.DateRules.WeekStart(daysOffset: int = 0)`	Trigger an event on the first day of each week plus an offset.
`self.DateRules.WeekStart(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`self.DateRules.WeekEnd(daysOffset: int = 0)`	Trigger an event on the last day of each week minus an offset.
`self.DateRules.WeekEnd(symbol: Symbol, daysOffset: int = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`self. DateRules.Today`	Trigger an event once today.
`self. DateRules.Tomorrow`	Trigger an event once tomorrow.

Time Rules

The following table describes the supported `TimeRules` :

Member	Description
`self.TimeRules.SetDefaultTimeZone(time_zone: DateTimeZone)`	Sets the time zone for the `TimeRules` object used in all methods in this table, except when a different time zone is given. The default time zone is the algorithm time zone .
`self.TimeRules.AfterMarketOpen(symbol: Symbol, minutesAfterOpen: float = 0, extendedMarketOpen: bool = False)`	Trigger an event a few minutes after market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`self.TimeRules.BeforeMarketClose(symbol: Symbol, minutesBeforeClose: float = 0, extendedMarketOpen: bool = False)`	Trigger an event a few minutes before market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`self.TimeRules.Every(interval: timedelta)`	Trigger an event every period interval starting at midnight.
`self. TimeRules.Now`	Trigger an event at the current time of day.
`self. TimeRules.Midnight`	Trigger an event at midnight.
`self. TimeRules.Noon`	Trigger an event at noon.
`self.TimeRules.At(hour: int, minute: int, second: int = 0)`	Trigger an event at a specific time of day (e.g. 13:10).
`self.TimeRules.At(hour: int, minute: int, second: int, time_zone: DateTimeZone)`	Trigger an event at a specific time of day in the given time zone (e.g. 13:10 UTC).

Examples

PY

```
# Select the universe on a specific date at a specific time
self.AddUniverseSelection(ScheduledUniverseSelectionModel(
    self.DateRules.On(2013, 10, 7),
    self.TimeRules.At(13, 0),
    self.select_symbols))

# Select the universe 10 minutes after SPY starts trading each day
self.AddUniverseSelection(ScheduledUniverseSelectionModel(
    self.DateRules.EveryDay("SPY"),
    self.TimeRules.AfterMarketOpen("SPY", 10),
    self.select_symbols))

# Select the universe 10 minutes before SPY stops trading each day
self.AddUniverseSelection(ScheduledUniverseSelectionModel(
    self.DateRules.EveryDay("SPY"),
    self.TimeRules.BeforeMarketClose("SPY", 10),
    self.select_symbols))

# Select the universe on Monday and Friday at noon
self.AddUniverseSelection(ScheduledUniverseSelectionModel(
    self.DateRules.Every(DayOfWeek.Monday, DayOfWeek.Friday),
    self.TimeRules.At(12, 0),
    self.select_symbols))

# Select the universe now
self.AddUniverseSelection(ScheduledUniverseSelectionModel(
    self.DateRules.Today,
```

```
        self.TimeRules.Now,
        self.select_symbols))

    # Select the universe tomorrow at midnight
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.Tomorrow,
        self.TimeRules.Midnight,
        self.select_symbols))

    # Select the universe today at noon
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.Today,
        self.TimeRules.Noon,
        self.select_symbols))

    # Select the universe every 10 minutes on everyday
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.EveryDay(),
        self.TimeRules.Every(timedelta(minutes=10)),
        self.select_symbols))

    # Select the universe at the market open on the first day of the month the SPY starts trading
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.MonthStart("SPY"),
        self.TimeRules.AfterMarketOpen("SPY"),
        self.select_symbols))

    # Select the universe at the market close on the last day of the month the SPY trades
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.MonthEnd("SPY"),
        self.TimeRules.BeforeMarketClose("SPY"),
        self.select_symbols))

    # Select the universe 5 minutes after SPY starts trading each week
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.WeekStart("SPY"),
        self.TimeRules.AfterMarketOpen("SPY", 5),
        self.select_symbols))

    # Select the universe 5 minutes before the SPY stops trading each week
    self.AddUniverseSelection(ScheduledUniverseSelectionModel(
        self.DateRules.WeekEnd("SPY"),
        self.TimeRules.BeforeMarketClose("SPY", 5),
        self.select_symbols))


    # Define a selection function that returns Symbol objects
    def select_symbols(self, date_time: datetime) -> List[Symbol]:
        tickers = ['SPY', 'AAPL', 'IBM']
        return [Symbol.Create(ticker, SecurityType.Equity, Market.USA)
            for ticker in tickers]
```

Demonstration Algorithms
ScheduledUniverseSelectionModelRegressionAlgorithm.py Python

Universe Selection

Futures Universes

Introduction

A Future Universe Selection model selects contracts for a set of Futures.

Future Universe Selection

The `FutureUniverseSelectionModel` selects all the contracts for a set of Futures you specify. To use this model, provide a `refreshInterval` and a selector function. The `refreshInterval` defines how frequently LEAN calls the selector function. The selector function receives a `datetime` object that represents the current Coordinated Universal Time (UTC) and returns a list of `Symbol` objects. The `Symbol` objects you return from the selector function are the Futures of the universe.

```
PY
from Selection.FutureUniverseSelectionModel import FutureUniverseSelectionModel

self.AddUniverseSelection(FutureUniverseSelectionModel(refreshInterval, futureChainSymbolSelector))
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`refreshInterval`	`timedelta`	Time interval between universe refreshes	
`futureChainSymbolSelector`	`Callable[[datetime], List[Symbol]]`	A function that selects the Future symbols for a given Coordinated Universal Time (UTC). To view the supported assets in the US Futures dataset, see Supported Assets .	
`universeSettings`	`UniverseSettings`	Universe settings define attributes of created subscriptions, such as their resolution and the minimum time in universe before they can be removed	`None`

If you don't provide a `universeSettings` argument, the `algorithm.UniverseSettings` is used by default.

```py
from Selection.FutureUniverseSelectionModel import FutureUniverseSelectionModel

def Initialize(self) -> None:
    universe = FutureUniverseSelectionModel(timedelta(days=1), self.select_future_chain_symbols)
    self.SetUniverseSelection(universe)

def select_future_chain_symbols(self, utc_time: datetime) -> List[Symbol]:
    return [ Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
             Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX) ]
```

This model uses the default Future contract filter, which doesn't select any Futures contracts. To use a different filter, subclass the `FutureUniverseSelectionModel` and define a `Filter` method. The `Filter` method accepts and returns a `FutureFilterUniverse` object to select the Futures contracts. The following table describes the filter methods of the `FutureFilterUniverse` class:

Method	Description
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weekly contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

Depending on how you define the contract filter, LEAN may call it once a day or at every time step.

```py
from Selection.FutureUniverseSelectionModel import FutureUniverseSelectionModel

# In Initialize
self.AddUniverseSelection(FrontMonthFutureUniverseSelectionModel())

# Outside of the algorithm class
class FrontMonthFutureUniverseSelectionModel(FutureUniverseSelectionModel):
    def __init__(self) -> None:
        super().__init__(timedelta(1), self.select_future_chain_symbols)

    def select_future_chain_symbols(self, utc_time: datetime) -> List[Symbol]:
        return [
            Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
            Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)
        ]

    def Filter(self, filter: FutureFilterUniverse) -> FutureFilterUniverse:
        return filter.FrontMonth().OnlyApplyFilterAtMarketOpen()
```

Some of the preceding filter methods only set an internal enumeration in the `FutureFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `FutureFilterUniverse` .

The `AddUniverseSelection` method doesn't return a `Future` object like the AddFuture method. The `Future` object contains `Symbol` and `Mapped` properties, which reference the continuous contract and the currently selected contract in the continuous contract series, respectively. To get the `Future` object, define the `OnSecuritiesChanged` method in your algorithm class or framework models and check the result of the `IsCanonical` method.

```py
def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
    for security in changes.AddedSecurities:
        if security.Symbol.IsCanonical():
            self.future = security
```

To view the implementation of this model, see the LEAN GitHub repository .

Open Interest Future Universe Selection

The `OpenInterestFutureUniverseSelectionModel` is an extension of the `FutureUniverseSelectionModel` that selects the contract with the greatest open interest on a daily basis.

```py
self.AddUniverseSelection(OpenInterestFutureUniverseSelectionModel(algorithm, futureChainSymbolSelector));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
algorithm	IAlgorithm	Algorithm	
futureChainSymbolSelector	Callable[[datetime], List[Symbol]]	A function that selects the Future symbols for a given Coordinated Universal Time (UTC). To view the supported assets in the US Futures dataset, see Supported Assets .	
chainContractsLookupLimit	int/NoneType	Limit on how many contracts to query for open interest	6
resultsLimit	int/NoneType	Limit on how many contracts will be part of the universe	1

```py
def Initialize(self):
    symbols = [
        Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
        Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)
    ]
    universe = OpenInterestFutureUniverseSelectionModel(self, lambda utc_time: symbols)
    self.AddUniverseSelection(universe)
```

To move the selection functions outside of the algorithm class, create a universe selection model that inherits the `OpenInterestFutureUniverseSelectionModel` class.

```py
# In Initialize
self.AddUniverseSelection(GoldOpenInterestFutureUniverseSelectionModel(self))

# Outside of the algorithm class
class GoldOpenInterestFutureUniverseSelectionModel(OpenInterestFutureUniverseSelectionModel):
    def __init__(self, algorithm: QCAlgorithm, chainContractsLookupLimit: int = 6, resultsLimit: int = 1):
        super().__init__(algorithm, self.select_future_chain_symbols, chainContractsLookupLimit,
resultsLimit)

    def select_future_chain_symbols(self, utcTime: datetime) -> List[Symbol]:
        return [Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)]
```

The `AddUniverseSelection` method doesn't return a `Future` object like the AddFuture method. The `Future` object contains `Symbol` and `Mapped` properties, which reference the continuous contract and the currently selected contract in the continuous contract series, respectively. To get the `Future` object, define the `OnSecuritiesChanged` method in your algorithm class or framework models and check the result of the `IsCanonical` method.

```py
def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
    for security in changes.AddedSecurities:
        if security.Symbol.IsCanonical():
            self.future = security
```

To view the implementation of this model, see the LEAN GitHub repository .

Universe Selection

Options Universes

Introduction

An Option Universe Selection model selects contracts for a set of Options.

Options Universe Selection

The `OptionUniverseSelectionModel` selects all the available contracts for the Equity Options, Index Options, and Future Options you specify. To use this model, provide a `refreshInterval` and a selector function. The `refreshInterval` defines how frequently LEAN calls the selector function. The selector function receives a `datetime` object that represents the current Coordinated Universal Time (UTC) and returns a list of `Symbol` objects. The `Symbol` objects you return from the selector function are the Options of the universe.

```
PY
from Selection.OptionUniverseSelectionModel import OptionUniverseSelectionModel

self.AddUniverseSelection(OptionUniverseSelectionModel(refreshInterval, optionChainSymbolSelector))
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`refreshInterval`	`timedelta`	Time interval between universe refreshes	
`optionChainSymbolSelector`	`Callable[[datetime], List[Symbol]]`	A function that selects the Option symbols	
`universeSettings`	`UniverseSettings`	Universe settings define attributes of created subscriptions, such as their resolution and the minimum time in universe before they can be removed	`None`

If you don't provide a `universeSettings` argument, the `algorithm.UniverseSettings` is used by default.

```py
from Selection.OptionUniverseSelectionModel import OptionUniverseSelectionModel

def Initialize(self) -> None:
    universe = OptionUniverseSelectionModel(timedelta(days=1), self.select_option_chain_symbols)
    self.SetUniverseSelection(universe)

def select_option_chain_symbols(self, utc_time: datetime) -> List[Symbol]:
    # Equity Options example:
    #tickers = ["SPY", "QQQ", "TLT"]
    #return [Symbol.Create(ticker, SecurityType.Option, Market.USA) for ticker in tickers]

    # Index Options example:
    #tickers = ["VIX", "SPX"]
    #return [Symbol.Create(ticker, SecurityType.IndexOption, Market.USA) for ticker in tickers]

    # Future Options example:
    future_symbol = Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME)
    future_contract_symbols = self.FutureChainProvider.GetFutureContractList(future_symbol, self.Time)
    return [Symbol.CreateCanonicalOption(symbol) for symbol in future_contract_symbols]
```

This model uses the default Option filter, which selects the Option contracts with the following characteristics at every time step:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 31 days

To use a different filter for the contracts, subclass the `OptionUniverseSelectionModel` and define a `Filter` method. The `Filter` method accepts and returns an `OptionFilterUniverse` object to select the Option contracts. The following table describes the methods of the `OptionFilterUniverse` class:

Method	Description
`Strikes(minStrike: int, maxStrike: int)`	Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price
`CallsOnly()`	Selects call contracts
`PutsOnly()`	Selects put contracts
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weeklys contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

Depending on how you define the contract filter, LEAN may call it once a day or at every time step.

To move the selection functions outside of the algorithm class, create a universe selection model that inherits the `OptionUniverseSelectionModel` class.

```
PY
from Selection.OptionUniverseSelectionModel import OptionUniverseSelectionModel

# In Initialize
self.AddUniverseSelection(EarliestExpiringAtTheMoneyCallOptionUniverseSelectionModel(self))

# Outside of the algorithm class
class EarliestExpiringAtTheMoneyCallOptionUniverseSelectionModel(OptionUniverseSelectionModel):
    def __init__(self, algorithm):
        self.algo = algorithm
        super().__init__(timedelta(1), self.select_option_chain_symbols)

    def select_option_chain_symbols(self, utc_time: datetime) -> List[Symbol]:
        # Equity Options example:
        #tickers = ["SPY", "QQQ", "TLT"]
        #return [Symbol.Create(ticker, SecurityType.Option, Market.USA) for ticker in tickers]

        # Index Options example:
        #tickers = ["VIX", "SPX"]
        #return [Symbol.Create(ticker, SecurityType.IndexOption, Market.USA) for ticker in tickers]

        # Future Options example:
        future_symbol = Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME)
        future_contract_symbols = self.algo.FutureChainProvider.GetFutureContractList(future_symbol,
self.algo.Time)
        return [Symbol.CreateCanonicalOption(symbol) for symbol in future_contract_symbols]

    def Filter(self, option_filter_universe: OptionFilterUniverse) -> OptionFilterUniverse:
        return option_filter_universe.Strikes(-1, -1).Expiration(0,
7).CallsOnly().OnlyApplyFilterAtMarketOpen()
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

To override the default pricing model of the Options, set a pricing model in a security initializer.

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

To view the implementation of this model, see the LEAN GitHub repository .

Option Chained Universe Selection

An Option chained universe subscribes to Option contracts on the constituents of a US Equity universe .

```
PY
self.AddUniverseOptions(universe, optionFilter)
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`universe`	`Universe`	The universe to chain onto the Option Universe Selection model	
`optionFilter`	`Callable[[OptionFilterUniverse], OptionFilterUniverse]`	The Option filter universe to use	

The `optionFilter` function receives and returns an `OptionFilterUniverse` to select the Option contracts. The following table describes the methods of the `OptionFilterUniverse` class:

Method	Description
`Strikes(minStrike: int, maxStrike: int)`	Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price
`CallsOnly()`	Selects call contracts
`PutsOnly()`	Selects put contracts
`StandardsOnly()`	Selects standard contracts
`IncludeWeeklys()`	Selects non-standard weeklys contracts
`WeeklysOnly()`	Selects weekly contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(minExpiry: timedelta, maxExpiry: timedelta)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(minExpiryDays: int, maxExpiryDays: int)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(contracts: List[Symbol])`	Selects a list of contracts
`Contracts(contractSelector: Callable[[List[Symbol]], List[Symbol]])`	Selects contracts that a selector function selects
`OnlyApplyFilterAtMarketOpen()`	Instructs the engine to only filter contracts on the first time step of each market day

PY

```
def Initialize(self) -> None:
    universe = self.AddUniverse(self.Universe.DollarVolume.Top(10))
    self.AddUniverseOptions(universe, self.OptionFilterFunction)

def OptionFilterFunction(self, option_filter_universe: OptionFilterUniverse) -> OptionFilterUniverse:
    return option_filter_universe.Strikes(-2, +2).FrontMonth().CallsOnly()
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

To override the default pricing model of the Options, set a pricing model in a security initializer.

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

To view the implementation of this model, see the LEAN GitHub repository .

Algorithm Framework

Alpha

Alpha

Key Concepts

Introduction

The Alpha model predicts market trends and signals the best moments to trade. These signals, or `Insight` objects, contain the `Direction` , `Magnitude` , and `Confidence` of a market prediction and the suggested portfolio `Weight` . You should generate insights on the set of assets provided by the Universe Selection model and only generate them when your predictions change.

Add Models

To add an Alpha model, in the `Initialize` method, call the `AddAlpha` method.

```
self.AddAlpha(EmaCrossAlphaModel())
```

PY

To view all the pre-built Alpha models, see Supported Models .

Multi-Alpha Algorithms

You can add multiple Alpha models to a single algorithm and generate Insight objects with all of them.

```
self.AddAlpha(RsiAlphaModel())
self.AddAlpha(EmaCrossAlphaModel())
```

PY

Each Alpha model has a unique name. The Insight objects they generate are automatically named according to the Alpha model name.

If you add multiple Alpha models, each alpha model receives the current slice in the order that you add the Alphas. The combined stream of Insight objects is passed to the Portfolio Construction model that defines how the Insight collection is combined. If you have a hybrid algorithm, the combined stream of Insight objects is passed to the event handler .

Model Structure

Alpha models should extend the `AlphaModel` class. Extensions of the `AlphaModel` class must implement the `Update` method, which receives a Slice object and returns an array of `Insight` objects. Extensions should also implement the `OnSecuritiesChanged` method to track security changes in the universe.

```py
# Algorithm framework model that produces insights
class MyAlphaModel(AlphaModel):

    def Update(self, algorithm: QCAlgorithm, data: Slice) -> List[Insight]:
        # Updates this Alpha model with the latest data from the algorithm.
        # This is called each time the algorithm receives data for subscribed securities
        # Generate insights on the securities in the universe.
        insights = []
        return insights

    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        # Security additions and removals are pushed here.
        # This can be used for setting up algorithm state.
        # changes.AddedSecurities
        # changes.RemovedSecurities
        pass
```

Method Parameters

The `algorithm` parameter that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it. To access members of your algorithm object, either use a global variable or pass an algorithm reference to Alpha model constructor. Both of these approaches violate the separation of concerns principle.

The `data` parameter contains the latest data available to the algorithm.

The `changes` parameter contains the universe changes.

Model Names

To ensure your Alpha model is compatible with all Portfolio Construction models , assign a unique name to your Alpha model. Some Portfolio Construction models can combine multiple Alpha models together, so it can be important to distinguish between the Alpha models.

```py
class RsiAlphaModel(AlphaModel):
    Name = "RsiAlphaModel"
```

By default, LEAN uses the string representation of a Guid object to set the Alpha model name. An example default name is "0f8fad5b-d9cb-469f-a165-70867728950e". This default name is different for every backtest you run and every live algorithm you deploy. For consistent behavior, manually set the Alpha model name.

Example

To view a full example of an `AlphaModel` subclass, see the ConstantAlphaModel in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Alpha model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Alpha model.

```
class MyAlphaModel(AlphaModel):
    symbol_data_by_symbol = {}

    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        for security in changes.AddedSecurities:
            self.symbol_data_by_symbol[security.Symbol] = SymbolData(algorithm, security.Symbol)

        for security in changes.RemovedSecurities:
            if security.Symbol in self.symbol_data_by_symbol:
                symbol_data = self.symbol_data_by_symbol.pop(security.Symbol, None)
                if symbol_data:
                    symbol_data.dispose()

class SymbolData:
    def __init__(self, algorithm, symbol):
        self.algorithm = algorithm
        self.symbol = symbol

        self.indicator = SimpleMovingAverage(20)
        self.consolidator = TradeBarConsolidator(1)
        algorithm.SubscriptionManager.AddConsolidator(symbol, self.consolidator)
        algorithm.RegisterIndicator(symbol, self.indicator, self.consolidator)
        algorithm.WarmUpIndicator(self.symbol, self.indicator)

    def dispose(self):
        self.algorithm.SubscriptionManager.RemoveConsolidator(self.symbol, self.consolidator)
```

Insights

An Insight is a single prediction for an asset. The `Update` method returns an array of Insight objects. You can think of these as actionable trading signals, indicating the asset direction, magnitude, and confidence in the near future. All insights can take a weight parameter to set the desired weighting for the insight. If you change the cash of the algorithm, it will affect the orders, but not necessarily the insights.

The Portfolio Construction model consumes the Insight objects from the Alpha model. It's up to the Portfolio Construction model to define how Insight objects are converted into `PortfolioTarget` objects. In the pre-built Portfolio Construction models, down insights translate to short-biased trades, up insights translate to long-biased trades, and flat insights usually close open positions, but this doesn't have to be the case in custom Portfolio Construction models .

Insight Properties

`Insight` objects have the following properties:

Create Insights

To create an `Insight` , call the `Insight` constructor.

```
# Insight(symbol, period, type, direction, magnitude=None, confidence=None, sourceModel=None, weight=None)
insight = Insight("IBM", timedelta(minutes=20), InsightType.Price, InsightDirection.Up)
```

In the `Insight` constructor, the `period` argument can be a `timedelta` object or a function that receives a `datetime` object and returns the expiry `datetime` . Some of the constructor arguments are optional to create the `Insight` object, but some of the Portfolio Construction models may require these properties. To check which `Insight` properties a Portfolio Construction model requires, see Supported Models .

You can also use the helper method to create Insight objects of the Price type.

```
# Insight.Price(symbol, period, direction, magnitude=None, confidence=None, sourceModel=None, weight=None)
insight = Insight.Price("IBM", timedelta(minutes = 20), InsightDirection.Up)

# Insight.Price(symbol, resolution, barCount, direction, magnitude=None, confidence=None,
sourceModel=None, weight=None)
insight = Insight.Price("IBM", Resolution.Minute, 20, InsightDirection.Up)
```

In the `Price` method, the `period` argument can be a `timedelta` object, a `datetime` object, or a function that receives a `datetime` object and returns the expiry `datetime` .

Group Insights

Sometimes an algorithm's performance relies on multiple insights being traded together, like in pairs trading and options straddles. These types insights should be grouped. Insight groups signal to the Execution model that the insights need to be acted on as a single unit to maximize the alpha created.

To mark insights as a group, call the `Insight.Group` method.

```
return Insight.Group([insight1, insight2,insight3])
```

Cancel Insights

If an Alpha model in your algorithm emits an Insight to enter a position but it determines the trading opportunity has pre-maturely ended, you should cancel the Insight. For example, say you want your algorithm to enter a long position in a security when its Relative Strength Index (RSI) moves below 20 and then liquidate the position when the security's RSI moves above 30. If you emit an Insight that has a duration of 30 days when the RSI moves below 20 but the RSI moves above 30 in 10 days, you should cancel the Insight.

To cancel an Insight, call its `Cancel` / `Expire` method with the algorithm's Coordinated Universal Time (UTC).

```
self.insight.Cancel(algorithm.UtcTime)
```

If you don't have a reference to the Insight you want to cancel, get it from the InsightManager .

When you cancel an insight, it's `CloseTimeUtc` property is set to one second into the past.

Stop Loss Orders Workaround

In some cases, if you add a Risk Management model that uses stop loss logic, the Risk Management model generates `PortfolioTarget` objects with a 0 quantity to make the Execution model liquidate your positions, but then the Portfolio Construction model generates `PortfolioTarget` objects with a non-zero quantity to make the Execution model re-enter the same position. This issue can occur if your Portfolio Construction model rebalances and your Alpha model still has an active insight for the liquidated securities. To avoid this issue, add the stop loss order logic to the Alpha model. When the stop loss is hit, emit a flat insight for the security. Note that this is a workaround, but it violates the separation of concerns principle since the Alpha Model shouldn't react to open positions.

Universe Timing Considerations

If the Alpha model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Alpha

Supported Models

Introduction

This page describes the pre-built Alpha models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Alpha model .

Null Model

The `NullAlphaModel` doesn't emit any insights. It's the default Alpha model.

```
self.AddAlpha(NullAlphaModel())
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Constant Model

The `ConstantAlphaModel` always returns the same insight for each security.

```
self.AddAlpha(ConstantAlphaModel(type, direction, period))
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`type`	`InsightType`	The type of insight	
`direction`	`InsightDirection`	The direction of the insight	
`period`	`timedelta`	The period over which the insight will come to fruition	
`magnitude`	`float/NoneType`	The predicted change in magnitude as a +/- percentage	`None`
`confidence`	`float/NoneType`	The confidence in the insight	`None`
`weight`	`float/NoneType`	The portfolio weight of the insights	`None`

To view the implementation of this model, see the LEAN GitHub repository .

Historical Returns Model

The `HistoricalReturnsAlphaModel` buys securities that have a positive trailing return and sells securities that have a negative trailing return. It sets the magnitude of the Insight objects to the trailing rate of change.

```
self.AddAlpha(HistoricalReturnsAlphaModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lookback`	`int`	Historical return lookback period	1
`resolution`	`Resolution`	The resolution of historical data	`Resolution.Daily`

This model cancels all the active insights it has emit for a security when the security has a 0% historical return.

To view the implementation of this model, see the LEAN GitHub repository .

EMA Cross Model

The `EmaCrossAlphaModel` uses an exponential moving average (EMA) cross to create insights. When the fast EMA crosses above the slow EMA, it emits up insights. When the fast EMA crosses below the slow EMA, it emits down insights. It sets the duration of Insight objects to be the product of the `resolution` and `fastPeriod` arguments.

```
self.AddAlpha(EmaCrossAlphaModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fastPeriod`	`int`	The fast EMA period	12
`slowPeriod`	`int`	The slow EMA period	26
`resolution`	`Resolution`	The resolution of data sent into the EMA indicators	`Resolution.Daily`

To view the implementation of this model, see the LEAN GitHub repository .

MACD Model

The `MacdAlphaModel` emits insights based on moving average convergence divergence (MACD) crossovers. If the MACD signal line is 1% above the security price, the model emits an up insight. If the MACD signal line is 1% below the security price, the model emits a down insight. If the MACD signal line is within 1% of the security price, the model cancels all the active insights it has emitted for the security.

```
self.AddAlpha(MacdAlphaModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fastPeriod`	`int`	The MACD fast period	12
`slowPeriod`	`int`	The MACD slow period	26
`signalPeriod`	`int`	The smoothing period for the MACD signal	9
`movingAverageType`	`MovingAverageType`	The type of moving average to use in the MACD	`MovingAverageType.Exponential`
`resolution`	`Resolution`	The resolution of data sent into the MACD indicator	`Resolution.Daily`

To view the implementation of this model, see the LEAN GitHub repository .

RSI Model

The `RsiAlphaModel` generates insights based on the relative strength index (RSI) indicator values. When the RSI value passes above 70, the model emits a down insight. When the RSI value passes below 30, the model emits an up insight. The model uses the Wilder moving average type and sets the duration of Insight objects to be the product of the `resolution` and `period` arguments.

```
self.AddAlpha(RsiAlphaModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`period`	`int`	The RSI indicator period	14
`resolution`	`Resolution`	The resolution of data sent into the RSI indicator	`Resolution.Daily`

To view the implementation of this model, see the LEAN GitHub repository .

Base Pairs Trading Model

The `BasePairsTradingAlphaModel` analyzes every possible pair combination from securities that the Universe Selection model selects. This model calculates a ratio between the two securities by dividing their historical prices over a lookback window. It then calculates the mean of this ratio by taking the 500-period EMA of the quotient. When the ratio

diverges far enough from the mean ratio, this model emits generates alternating long ratio/short ratio insights emitted as a group to capture the reversion of the ratio.

```
self.AddAlpha(BasePairsTradingAlphaModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lookback`	`int`	Lookback period of the analysis	1
`resolution`	`Resolution`	Analysis resolution	`Resolution.Daily`
`threshold`	`float`	The percent [0, 100] deviation of the ratio from the mean before emitting an insight	1

To view the implementation of this model, see the LEAN GitHub repository .

Pearson Correlation Pairs Trading Model

The `PearsonCorrelationPairsTradingAlphaModel` ranks every pair combination by its Pearson correlation coefficient and trades the pair with the highest correlation. This model follows the same insight logic as the `BasePairsTradingModel` .

```
self.AddAlpha(PearsonCorrelationPairsTradingAlphaModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lookback`	`int`	Lookback period of the analysis	15
`resolution`	`Resolution`	Analysis resolution	`Resolution.Minute`
`threshold`	`float`	The percent [0, 100] deviation of the ratio from the mean before emitting an insight	1
`minimumCorrelation`	`double float`	The minimum correlation to consider a tradable pair	0.5

To view the implementation of this model, see the LEAN GitHub repository .

Algorithm Framework

Portfolio Construction

Portfolio Construction

Key Concepts

Introduction

The Portfolio Construction model receives `Insight` objects from the Alpha model and creates `PortfolioTarget` objects for the Risk Management model. A `PortfolioTarget` provides the number of units of an asset to hold.

Set Models

To set a Portfolio Construction model, in the `Initialize` method, call the `SetPortfolioConstruction` method.

```
self.SetPortfolioConstruction(EqualWeightingPortfolioConstructionModel())
```

To view all the pre-built Portfolio Construction models, see Supported Models .

Model Structure

Portfolio Construction models should extend the `PortfolioConstructionModel` class or one of the supported models . Extensions of the `PortfolioConstructionModel` class should implement the `CreateTargets` method, which receives an array of `Insight` objects from the Alpha model at every time step and returns an array of `PortfolioTarget` objects. The Portfolio Construction model seeks to answer the question, "how many units should I buy based on the insight predictions I've been presented?".

If you don't override the `CreateTargets` method, the base class implementation calls the model's `IsRebalanceDue` , `DetermineTargetPercent` , and `GetTargetInsights` helper methods. The `GetTargetInsights` method, in turn, calls the model's `ShouldCreateTargetForInsight` method. You can override any of these helper methods. If you don't override the `CreateTargets` method from the `PortfolioConstructionModel` class, your class must at least override the `DetermineTargetPercent` method.

```
 # Portfolio construction scaffolding class; basic method args.
class MyPortfolioConstructionModel(PortfolioConstructionModel):
    # Create list of PortfolioTarget objects from Insights
    def CreateTargets(self, algorithm: QCAlgorithm, insights: List[Insight]) -> List[PortfolioTarget]:
        return super().CreateTargets(algorithm, insights)

    # Determines if the portfolio should rebalance based on the provided rebalancing func
    def IsRebalanceDue(self, insights: List[Insight], algorithmUtc: datetime) -> bool:
        return super().IsRebalanceDue(insights, algorithmUtc)

    # Determines the target percent for each insight
    def DetermineTargetPercent(self, activeInsights: List[Insight]) -> Dict[Insight, float]:
        return {}

    # Gets the target insights to calculate a portfolio target percent for, they will be piped to
DetermineTargetPercent()
    def GetTargetInsights(self) -> List[Insight]:
        return super().GetTargetInsights()

    # Determine if the portfolio construction model should create a target for this insight
    def ShouldCreateTargetForInsight(self, insight: Insight) -> bool:
        return super().ShouldCreateTargetForInsight(insight)

    # Security change details
    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        super().OnSecuritiesChanged(algorithm, changes)
```

The Portfolio Construction model should remove expired insights from the Insight Manager . The `CreateTargets` definition of the base `PortfolioConstructionModel` class already removes them during each rebalance. Therefore, if you override the `CreateTargets` method and don't call the `CreateTargets` definition of the base class, your new method definition should remove expired insights from the Insight Manager.

The model should also remove all a security's insights from the Insight Manager when the security is removed from the universe . The `OnSecuritiesChanged` definition of the base `PortfolioConstructionModel` class already does this. Therefore, if you override the `OnSecuritiesChanged` method and don't call the `OnSecuritiesChanged` definition of the base class, your new method definition should remove the security's insights from the Insight Manager.

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

You may use the `PortfolioBias` enumeration in the definition of Portfolio Construction model methods. The `PortfolioBias` enumeration has the following members:

To view a full example of a `PortfolioConstructionModel` subclass, see the EqualWeightingPortfolioConstructionModel in the LEAN GitHub repository.

Multi-Alpha Algorithms

If you add multiple Alpha models , each Alpha model receives the current slice in the order that you add the Alphas. The combined stream of Insight objects is passed to the Portfolio Construction model.

Each Portfolio Construction model has a unique method to combine Insight objects. The base `PortfolioConstructionModel` that most PCM's inherit from doesn't combine information from Insight objects with the same `Symbol` - but just gets the most recent active insight. To combine the active insights differently, override the `GetTargetInsights` , and return all active insights. The `DetermineTargetPercent` method implements the combination criteria and determines the target for each `Symbol` .

```
class MultipleAlphaPortfolioConstructionModel(PortfolioConstructionModel):
    def GetTargetInsights(self) -> List[Insight]:
        return list(self.InsightCollection.GetActiveInsights(self.Algorithm.UtcTime))

    def DetermineTargetPercent(self, activeInsights: List[Insight]) -> Dict[Insight, float]:
        return {}
```

Portfolio Targets

The Portfolio Construction model returns `PortfolioTarget` objects, which are passed to the Risk Management model.

To create a `PortfolioTarget` object based on a quantity, pass the `Symbol` and quantity to the `PortfolioTarget` constructor.

```
# Create a new portfolio target for 1200 IBM shares.
target = PortfolioTarget("IBM", 1200)
```

To create a `PortfolioTarget` object based on a portfolio weight, call the `Percent` method. This method is only available for margin accounts.

```
# Calculate target equivalent to 10% of portfolio value
target = PortfolioTarget.Percent(algorithm, "IBM", 0.1)
```

The `CreateTargets` method of your Portfolio Construction model must return an array of `PortfolioTarget` objects.

```
return [ PortfolioTarget("IBM", 1200) ]
```

Portfolio Target Collection

The `PortfolioTargetCollection` class is a helper class to manage `PortfolioTarget` objects. The class manages an internal dictionary that has the security `Symbol` as the key and a `PortfolioTarget` as the value.

Add Portfolio Targets

To add a `PortfolioTarget` to the `PortfolioTargetCollection` , call the `Add` method.

```
self.targets_collection.Add(portfolio_target)
```

To add a list of `PortfolioTarget` objects, call the `AddRange` method.

```
self.targets_collection.AddRange(portfolio_targets)
```

Check Membership

To check if a `PortfolioTarget` exists in the `PortfolioTargetCollection` , call the `Contains` method.

```
target_in_collection = self.targets_collection.Contains(portfolio_target)
```

To check if a Symbol exists in the `PortfolioTargetCollection` , call the `ContainsKey` method.

```
symbol_in_collection = self.targets_collection.ContainsKey(symbol)
```

To get all the Symbol objects, use the `Keys` property.

```
symbols = self.targets_collection.Keys
```

Access Portfolio Targets

To access the `PortfolioTarget` objects for a Symbol, index the `PortfolioTargetCollection` with the Symbol.

```
portfolio_target = self.targets_collection[symbol]
```

To iterate through the `PortfolioTargetCollection` , call the `GetEnumerator` method.

```
enumerator = self.targets_collection.GetEnumerator()
```

To get all the `PortfolioTarget` objects, use the `Values` property

```
portfolio_targets = self.targets_collection.Values
```

Order Portfolio Targets by Margin Impact

To get an enumerable where position reducing orders are executed first and the remaining orders are executed in decreasing order value, call the `OrderByMarginImpact` method.

```
for target in self.targets_collection.OrderByMarginImpact(algorithm):
    # Place order
```

This method won't return targets for securities that have no data yet. This method also won't return targets for which the sum of the current holdings and open orders quantity equals the target quantity.

Remove Portfolio Targets

To remove a `PortfolioTarget` from the `PortfolioTargetCollection` , call the `Remove` method.

```
remove_successful = self.targets_collection.Remove(symbol)
```

To remove all the `PortfolioTarget` objects, call the `Clear` method.

```
self.targets_collection.Clear()
```

To remove all the `PortfolioTarget` objects that have been fulfilled, call the `ClearFulfilled` method.

```
self.targets_collection.ClearFulfilled(algorithm)
```

Copy Portfolio Targets

To copy a subset of the `PortfolioTarget` objects in the `PortfolioTargetCollection` to an array, call the `CopyTo` method. The `arrayIndex` argument is the zero-based index in the array at which copying begins.

```
self.targets_collection.CopyTo(array, arrayIndex)
```

Rebalance Frequency

If you use a Portfolio Construction model that is a subclass of the `PortfolioConstructionModel` class, you can set the rebalancing frequency of the model with a function. The rebalancing function receives the Coordinated Universal Time (UTC) of the algorithm and should return the next rebalance UTC time or `None` . If the function returns `None` , the model doesn't rebalance unless the rebalance settings trigger a rebalance. For a full example of a custom rebalance function, see the PortfolioRebalanceOnCustomFuncRegressionAlgorithm .

If you use a Portfolio Construction model with the following characteristics, you can also set the rebalancing frequency of the model with a `timedelta` , `Resolution` , or DateRules :

- The model is a subclass of the `EqualWeightingPortfolioConstructionModel` class.
- The model constructor calls the `EqualWeightingPortfolioConstructionModel` constructor.
- The model doesn't override the `CreateTargets` method.

To check which of the pre-built Portfolio Construction models support this functionality, see Supported Models .

Rebalance Settings

By default, portfolio construction models create `PortfolioTarget` objects to rebalance the portfolio when any of the following events occur:

- The model's rebalance function signals it's time to rebalance
- The Alpha model emits new insights
- The universe changes

To disable rebalances when the Alpha model emits insights or when insights expire, set

`RebalancePortfolioOnInsightChanges` to false.

```
# In Initialize
self.Settings.RebalancePortfolioOnInsightChanges = False
```

To disable rebalances when security changes occur, set `RebalancePortfolioOnSecurityChanges` to false.

```
# In Initialize
self.Settings.RebalancePortfolioOnSecurityChanges = False
```

Portfolio Optimizer Structure

Some portfolio construction models contain an optimizer that accepts the historical returns of each security and returns a list of optimized portfolio weights. Portfolio optimizer models must implement the `IPortfolioOptimizer` interface, which has an `Optimize` method.

```
class MyPortfolioOptimizer:

    def Optimize(self, historicalReturns: pd.DataFrame, expectedReturns: pd.Series = None, covariance:
pd.DataFrame = None) -> pd.Series:
        # Create weights
        return weights
```

The following table describes the arguments the `Optimize` method accepts:

Argument	Data Type	Description	Default Value
`historicalReturns`	`DataFrame`	Matrix of annualized historical returns where each column represents a security and each row returns for the given date/time (size: K x N)	
`expectedReturns`	`Series`	Array of double with the portfolio annualized expected returns (size: K x 1)	`None`
`covariance`	`DataFrame`	Multi-dimensional array of double with the portfolio covariance of annualized returns (size: K x K)	`None`

The method should return a K x 1 array of double objects that represent the portfolio weights.

To view all the pre-built portfolio optimization algorithms, see Supported Optimizers .

To view a full example of an `IPortfolioOptimizer` implementation, see the MaximumSharpeRatioPortfolioOptimizer in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Portfolio Construction model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Portfolio Construction model.

```
PY
class MyPortfolioConstructionModel(PortfolioConstructionModel):
    symbol_data_by_symbol = {}

    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        super().OnSecuritiesChanged(algorithm, changes)
        for security in changes.AddedSecurities:
            self.symbol_data_by_symbol[security.Symbol] = SymbolData(security.Symbol)

        for security in changes.RemovedSecurities:
            if security.Symbol in self.symbol_data_by_symbol:
                self.symbol_data_by_symbol.pop(security.Symbol, None)

class SymbolData:
    def __init__(self, symbol):
        self.symbol = symbol
        # Store and manage Symbol-specific data
```

Universe Timing Considerations

If the Portfolio Construction model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Portfolio Construction

Supported Models

Introduction

This page describes the pre-built Portfolio Construction models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Portfolio Construction model .

Null Model

The `NullPortfolioConstructionModel` is the default Portfolio Construction model. It doesn't return any `PortfolioTarget` objects. It's useful if you need to analyze an Alpha model in isolation.

```
PY
self.SetPortfolioConstruction(NullPortfolioConstructionModel())
```

To view the implementation of this model, see the LEAN GitHub repository .

Equal Weighting Model

The `EqualWeightingPortfolioConstructionModel` assigns an equal share of the portfolio to the securities with active insights. This weighting scheme is useful for universe rotation based on simple portfolio strategies.

```
PY
self.SetPortfolioConstruction(EqualWeightingPortfolioConstructionModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Confidence Weighted Model

The `ConfidenceWeightedPortfolioConstructionModel` generates target portfolio weights based on the `Insight.Confidence` for the last Insight of each Symbol. If the Insight has a direction of `InsightDirection.Up` , the model generates long targets. If the Insight has a direction of `InsightDirection.Down` , the model generates short

targets. If the sum of all the last active Insight per Symbol is greater than 1, the model factors down each target percent holdings proportionally so the sum is 1. The model ignores `Insight` objects that have no `Confidence` value.

```
self.SetPortfolioConstruction(ConfidenceWeightedPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Insight Weighting Model

The `InsightWeightingPortfolioConstructionModel` generates target portfolio weights based on the `Insight.Weight` for the last Insight of each Symbol. If the Insight has a direction of `InsightDirection.Up` , the model generates long targets. If the Insight has a direction of `InsightDirection.Down` , the model generates short targets. If the sum of all the last active Insight per Symbol is greater than 1, the model factors down each target percent holdings proportionally so the sum is 1. The model takes the absolute value of the `Weight` of each `Insight` object and ignores `Insight` objects that have no `Weight` value.

```
self.SetPortfolioConstruction(InsightWeightingPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Sector Weighting Model

The `SectorWeightingPortfolioConstructionModel` generates target portfolio weights based on the `CompanyReference.IndustryTemplateCode` provided by the US Fundamental dataset . The target percent holdings of each sector is 1/S where S is the number of sectors and the target percent holdings of each security is 1/N where N is the number of securities of each sector. If the insight has a direction of `InsightDirection.Up` , the model generates long targets. If the insight has a direction of `InsightDirection.Down` , the model generates short targets. The model ignores `Insight` objects for Symbols that have no `CompanyReference.IndustryTemplateCode` .

```
self.SetPortfolioConstruction(SectorWeightingPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Accumulative Insight Model

The `AccumulativeInsightPortfolioConstructionModel` generates target portfolio weights based on all the active insights of a security. For each active Insight of direction `InsightDirection.Up` , it increases the position size by a fixed percent. For each active Insight of direction `InsightDirection.Down` , it decreases the position size by a fixed percent. For each active Insight of direction `InsightDirection.Flat` , it moves the position size towards 0 by a fixed percent.

```
self.SetPortfolioConstruction(AccumulativeInsightPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	Any of the following types: • `timedelta` • `Resolution` • DateRules • `None` • Callable[[datetime], datetime]	Rebalancing parameter. If it's a `timedelta` , `DateRules` or `Resolution` , it's converted into a function. If it's `None` , it's ignored. The function returns the next expected rebalance time for a given algorithm UTC DateTime. The function returns `None` if unknown, in which case the function will be called again in the next loop. If the function returns the current time, the portfolio rebalances.	`None`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`percent`	`PortfolioBias`	The percentage amount of the portfolio value to allocate to a single insight	0.03 (3%)

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Mean Variance Optimization Model

The `MeanVarianceOptimizationPortfolioConstructionModel` seeks to build a portfolio with the least volatility possible and achieve a target return.

```
self.SetPortfolioConstruction(MeanVarianceOptimizationPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`lookback`	`int`	Historical return lookback period	1
`period`	`int`	The time interval of history price to calculate the weight	63
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`
`targetReturn`	`float`	The target portfolio return	0.02 (2%)
`optimizer`	`IPortfolioOptimizer`	The portfolio optimization algorithm	`None`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency . If you don't provide an `optimizer` argument, the default one is the MinimumVariancePortfolioOptimizer with upper and lower weights that respect the `portfolioBias` .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Black Litterman Optimization Model

The `BlackLittermanOptimizationPortfolioConstructionModel` receives `Insight` objects from multiple Alphas and combines them into a single portfolio. These multiple sources of Alpha models can be seen as the "investor views" required in the classical model.

```
self.SetPortfolioConstruction(BlackLittermanOptimizationPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`lookback`	`int`	Historical return lookback period	1
`period`	`int`	The time interval of history price to calculate the weight	63
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`
`risk_free_rate`	`float`	The risk free rate	0.0
`delta`	`float`	The risk aversion coefficient of the market portfolio	2.5
`tau`	`float`	The model parameter indicating the uncertainty of the CAPM prior	0.05
`optimizer`	`IPortfolioOptimizer`	The portfolio optimization algorithm	`None`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency . If you don't provide an `optimizer` argument, the default one is the MinimumVariancePortfolioOptimizer with upper and lower weights that respect the `portfolioBias` .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Mean Reversion Model

The `MeanReversionPortfolioConstructionModel` implements an on-line portfolio selection technique, named "On-Line Moving Average Reversion" (OLMAR). The basic idea is to represent multi-period mean reversion as "Moving Average Reversion" (MAR), which explicitly predicts next price relatives using moving averages and then forms portfolios with online learning techniques.

```
self.SetPortfolioConstruction(MeanReversionPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`reversionThreshold`	`float`	Reversion threshold	1
`windowSize`	`int`	The window size of mean price	20
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Reference:

- Li, B., Hoi, S. C. (2012). On-line portfolio selection with moving average reversion. arXiv preprint arXiv:1206.4626 .

Risk Parity Model

The `RiskParityPortfolioConstructionModel` seeks to build a portfolio with the equal contribution of risk to the total portfolio risk from all assets.

```
self.SetPortfolioConstruction(RiskParityPortfolioConstructionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalance`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`lookback`	`int`	Historical return lookback period	1
`period`	`int`	The time interval of history price to calculate the weight	252
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`
`optimizer`	`IPortfolioOptimizer`	The portfolio optimization algorithm	`None`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency . If you don't provide an `optimizer` argument, the default one is the RiskParityPortfolioOptimizer .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

To view the implementation of this model, see the LEAN GitHub repository .

Portfolio Construction

Supported Optimizers

Introduction

This page describes the pre-built Portfolio Optimizer models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Portfolio Optimizer model .

Maximum Sharpe Ratio Optimizer

The `MaximumSharpeRatioPortfolioOptimizer` seeks to maximize the portfolio Sharpe Ratio .

```
optimizer = MaximumSharpeRatioPortfolioOptimizer()
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`minimum_weight`	`float`	The lower bounds on portfolio weights	-1
`maximum_weight`	`float`	The upper bounds on portfolio weights	1
`risk_free_rate`	`float`	The risk free rate	0

To view the implementation of this model, see the LEAN GitHub repository .

Minimum Variance Optimizer

The `MinimumVariancePortfolioOptimizer` seeks to minimize the portfolio variance and achieve a target return.

```
optimizer = MinimumVariancePortfolioOptimizer()
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`minimum_weight`	`float`	The lower bounds on portfolio weights	-1
`maximum_weight`	`float`	The upper bounds on portfolio weights	1
`target_return`	`float`	The target portfolio return	0.02 (2%)

To view the implementation of this model, see the LEAN GitHub repository .

Unconstrained Mean Variance Optimizer

The `UnconstrainedMeanVariancePortfolioOptimizer` seeks to find the optimal risk-adjusted portfolio that lies on the efficient frontier.

```
optimizer = UnconstrainedMeanVariancePortfolioOptimizer(historical_returns)
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`historicalReturns`	`DataFrame`	Matrix of historical returns where each column represents a security and each row returns for the given date/time (size: K x N)	
`expectedReturns`	`Series`	Array of double with the portfolio expected returns (size: K x 1)	`None`
`covariance`	`DataFrame`	Multi-dimensional array of double with the portfolio covariance of returns (size: K x K)	`None`

If you don't provide an argument for `expectedReturns` or `covariance` , it's calculated based on the `historicalReturns` .

To view the implementation of this model, see the LEAN GitHub repository .

Risk Parity Optimizer

The `RiskParityPortfolioOptimizer` seeks to equalize the individual risk contribution to the total portfolio risk from each asset.

```
optimizer = RiskParityPortfolioOptimizer()
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`minimum_weight`	`float`	The lower bounds on portfolio weights	1e-05
`maximum_weight`	`float`	The upper bounds on portfolio weights	`sys.float_info.max`

To view the implementation of this model, see the LEAN GitHub repository .

Algorithm Framework

Risk Management

Risk Management

Key Concepts

Introduction

The Risk Management model seeks to manage risk on the `PortfolioTarget` collection it receives from the Portfolio Construction model before the targets reach the Execution model. There are many creative ways to manage risk. Some examples of risk management include the following:

- *"Trailing Stop Risk Management Model"*
  Create and manage trailing stop-loss orders for open positions.
- *"Option Hedging Risk Management Model"*
  Purchase options to hedge large equity exposures.
- *"Sector Exposure Risk Management Model"*
  Reduce position sizes when overexposed to sectors or individual assets, keeping the portfolio within diversification requirements.
- *"Flash Crash Detection Risk Management Model"*
  Scan for strange market situations that might be precursors to a flash crash and attempt to protect the portfolio when they are detected.

Add Models

To set a Risk Management model, in the `Initialize` method, call the `AddRiskManagement` method.

```
self.AddRiskManagement(NullRiskManagementModel())
```

PY

To view all the pre-built Risk Management models, see Supported Models .

Multi-Model Algorithms

To add multiple Risk Management models, in the `Initialize` method, call the `AddRiskManagement` method multiple times.

```
self.AddRiskManagement(MaximumDrawdownPercentPerSecurity())
self.AddRiskManagement(MaximumSectorExposureRiskManagementModel())
```

PY

If you add multiple Risk Management models, the original collection of `PortfolioTarget` objects from the Portfolio Construction model is passed to the first Risk Management model. The risk-adjusted targets from the first Risk Management model are passed to the second Risk Management model. The process continues sequentially until all of the Risk Management models have had an opportunity to adjust the targets.

Model Structure

Risk Management models should extend the `RiskManagementModel` class. Extensions of the `RiskManagementModel` class must implement the `ManageRisk` method, which receives an array of `PortfolioTarget` objects from the Portfolio Construction model at every time step and should return an array of risk-adjusted `PortfolioTarget` objects. The method should only return the adjusted targets, not all of targets. If the method creates a `PortfolioTarget` object to liquidate a security, cancel the security's insights to avoid re-entering the position.

```
PY
class MyRiskManagementModel(RiskManagementModel):
    # Adjust the portfolio targets and return them. If no changes emit nothing.
    def ManageRisk(self, algorithm: QCAlgorithm, targets: List[PortfolioTarget]) -> List[PortfolioTarget]:
        return []

    # Optional: Be notified when securities change
    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        # Security additions and removals are pushed here.
        # This can be used for setting up algorithm state.
        # changes.AddedSecurities
        # changes.RemovedSecurities
        pass
```

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

To view a full example of a `RiskManagementModel` subclass, see the MaximumDrawdownPercentPerSecurity in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Risk Management model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Risk Management model.

```
PY
class MyRiskManagementModel(RiskManagementModel):
    symbol_data_by_symbol = {}

    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        super().OnSecuritiesChanged(algorithm, changes)
        for security in changes.AddedSecurities:
            self.symbol_data_by_symbol[security.Symbol] = SymbolData(security.Symbol)

        for security in changes.RemovedSecurities:
            if security.Symbol in self.symbol_data_by_symbol:
                self.symbol_data_by_symbol.pop(security.Symbol, None)

class SymbolData:
    def __init__(self, symbol):
        self.symbol = symbol
        # Store and manage Symbol-specific data
```

Portfolio Target Collection

The `PortfolioTargetCollection` class is a helper class to manage `PortfolioTarget` objects. The class manages an internal dictionary that has the security `Symbol` as the key and a `PortfolioTarget` as the value.

Add Portfolio Targets

To add a `PortfolioTarget` to the `PortfolioTargetCollection` , call the `Add` method.

```
self.targets_collection.Add(portfolio_target)
```

PY

To add a list of `PortfolioTarget` objects, call the `AddRange` method.

```
self.targets_collection.AddRange(portfolio_targets)
```

PY

Check Membership

To check if a `PortfolioTarget` exists in the `PortfolioTargetCollection` , call the `Contains` method.

```
target_in_collection = self.targets_collection.Contains(portfolio_target)
```

PY

To check if a Symbol exists in the `PortfolioTargetCollection` , call the `ContainsKey` method.

```
symbol_in_collection = self.targets_collection.ContainsKey(symbol)
```

PY

To get all the Symbol objects, use the `Keys` property.

```
symbols = self.targets_collection.Keys
```

PY

Access Portfolio Targets

To access the `PortfolioTarget` objects for a Symbol, index the `PortfolioTargetCollection` with the Symbol.

```
portfolio_target = self.targets_collection[symbol]
```

PY

To iterate through the `PortfolioTargetCollection` , call the `GetEnumerator` method.

```
enumerator = self.targets_collection.GetEnumerator()
```

PY

To get all the `PortfolioTarget` objects, use the `Values` property

```
portfolio_targets = self.targets_collection.Values
```

Order Portfolio Targets by Margin Impact

To get an enumerable where position reducing orders are executed first and the remaining orders are executed in decreasing order value, call the `OrderByMarginImpact` method.

```
for target in self.targets_collection.OrderByMarginImpact(algorithm):
    # Place order
```

This method won't return targets for securities that have no data yet. This method also won't return targets for which the sum of the current holdings and open orders quantity equals the target quantity.

Remove Portfolio Targets

To remove a `PortfolioTarget` from the `PortfolioTargetCollection` , call the `Remove` method.

```
remove_successful = self.targets_collection.Remove(symbol)
```

To remove all the `PortfolioTarget` objects, call the `Clear` method.

```
self.targets_collection.Clear()
```

To remove all the `PortfolioTarget` objects that have been fulfilled, call the `ClearFulfilled` method.

```
self.targets_collection.ClearFulfilled(algorithm)
```

Copy Portfolio Targets

To copy a subset of the `PortfolioTarget` objects in the `PortfolioTargetCollection` to an array, call the `CopyTo` method. The `arrayIndex` argument is the zero-based index in the array at which copying begins.

```
self.targets_collection.CopyTo(array, arrayIndex)
```

Universe Timing Considerations

If the Risk Management model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator

- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Risk Management

Supported Models

Introduction

This page describes the pre-built Risk Management models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Risk Management model .

Null Model

The `NullRiskManagementModel` is the default Risk Management model. It doesn't adjust any of the `PortfolioTarget` objects it receives from the Portfolio Construction model.

```
self.AddRiskManagement(NullRiskManagementModel())
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Maximum Security Drawdown Model

The `MaximumDrawdownPercentPerSecurity` model monitors the unrealized profit percentage of each security in the portfolio. When the percentage drops below a threshold relative to the opening price, it liquidates the position and cancels all insights in the Insight Manager that are for the security. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
self.AddRiskManagement(MaximumDrawdownPercentPerSecurity())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumDrawdownPercent`	`float`	The maximum percentage drawdown allowed for any single security holding	0.05 (5%)

To view the implementation of this model, see the LEAN GitHub repository .

Maximum Portfolio Drawdown Model

The `MaximumDrawdownPercentPortfolio` model monitors the portfolio drawdown . The drawdown can be relative to the starting portfolio value or the maximum portfolio value. When the portfolio value drops below a percentage threshold, the model liquidates the portfolio and cancels all insights in the Insight Manager . To liquidate the portfolio, the model

must receive at least 1 `PortfolioTarget` after the drawdown threshold is passed. After the portfolio is liquidated, the model resets. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
self.AddRiskManagement(MaximumDrawdownPercentPortfolio())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumDrawdownPercent`	`float`	Maximum spread accepted comparing to current price in percentage	0.05 (5%)
`isTrailing`	`bool`	If "false", the drawdown is relative to the starting value of the portfolio. If "true", the drawdown is relative the last maximum portfolio value	`False`

To view the implementation of this model, see the LEAN GitHub repository .

Maximum Unrealized Profit Model

The `MaximumUnrealizedProfitPercentPerSecurity` model monitors the unrealized profit of each security in the portfolio. When the unrealized profit exceeds a profit threshold, it liquidates the position and cancels all insights in the Insight Manager that are for the security. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
self.AddRiskManagement(MaximumUnrealizedProfitPercentPerSecurity())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumUnrealizedProfitPercent`	`float`	The maximum percentage unrealized profit allowed for any single security holding	0.05 (5%)

To view the implementation of this model, see the LEAN GitHub repository .

Maximum Sector Exposure Model

The `MaximumSectorExposureRiskManagementModel` limits the absolute portfolio exposure in a each industry sector to a predefined maximum percentage. If the absolute portfolio exposure exceeds the maximum percentage, the weight of each Equity in the sector is scaled down so the sector doesn't exceed the maximum percentage. This process requires

assets that are selected by Morningstar fine fundamental data . This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
self.AddRiskManagement(MaximumSectorExposureRiskManagementModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumSectorExposure`	`float`	The maximum exposure for any sector	0.2 (20%)

To view the implementation of this model, see the LEAN GitHub repository .

Trailing Stop Model

The `TrailingStopRiskManagementModel` monitors the drawdown of each security in the portfolio. When the peak-to-trough drawdown of the unrealized profit exceeds a threshold, it liquidates the position and cancels all insights in the Insight Manager that are for the security. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
self.AddRiskManagement(TrailingStopRiskManagementModel())
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumDrawdownPercent`	`float`	The maximum percentage drawdown allowed for algorithm portfolio compared with the highest unrealized profit	0.05 (5%)

To view the implementation of this model, see the LEAN GitHub repository .

Algorithm Framework

Execution

Execution

Key Concepts

Introduction

The Execution model receives an array of risk-adjusted `PortfolioTarget` objects from the Risk Management model and places trades in the market to satisfy the targets. The Execution model only receives updates to the portfolio target share counts. It doesn't necessarily receive all of the targets at once.

Set Models

To set an Execution model, in the `Initialize` method, call the `SetExecution` method.

```
self.SetExecution(ImmediateExecutionModel())
```

To view all the pre-built Execution models, see Supported Models .

Model Structure

Execution models should extend the `ExecutionModel` class. Extensions of the `ExecutionModel` must implement the `Execute` method, which receives an array of `PortfolioTarget` objects at every time step and is responsible for reaching the target portfolio as efficiently as possible. The Portfolio Construction model creates the `PortfolioTarget` objects, the Risk Management model may adjust them, and then the Execution model places the orders to fulfill them.

```
# Execution Model scaffolding structure example
class MyExecutionModel(ExecutionModel):

    # Fill the supplied portfolio targets efficiently
    def Execute(self, algorithm: QCAlgorithm, targets: List[PortfolioTarget]) -> None:
        pass

    # Optional: Securities changes event for handling new securities.
    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        # Security additions and removals are pushed here.
        # This can be used for setting up algorithm state.
        # changes.AddedSecurities
        # changes.RemovedSecurities
        pass
```

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

The following table describes the properties of the `PortfolioTarget` class that you may access in the Execution model:

Property	Data Type	Description
`Symbol`	`Symbol`	Asset to trade
`Quantity`	`float`	Number of units to hold

To view a full example of an `ExecutionModel` subclass, see the ImmediateExecutionModel in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Execution model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Execution model.

```
PY
class MyExecutionModel(ExecutionModel):
    symbol_data_by_symbol = {}

    def OnSecuritiesChanged(self, algorithm: QCAlgorithm, changes: SecurityChanges) -> None:
        for security in changes.AddedSecurities:
            self.symbol_data_by_symbol[security.Symbol] = SymbolData(security.Symbol)

        for security in changes.RemovedSecurities:
            if security.Symbol in self.symbol_data_by_symbol:
                self.symbol_data_by_symbol.pop(security.Symbol, None)

class SymbolData:
    def __init__(self, symbol):
        self.symbol = symbol
        # Store and manage Symbol-specific data
```

Portfolio Target Collection

The `PortfolioTargetCollection` class is a helper class to manage `PortfolioTarget` objects. The class manages an internal dictionary that has the security `Symbol` as the key and a `PortfolioTarget` as the value.

Add Portfolio Targets

To add a `PortfolioTarget` to the `PortfolioTargetCollection` , call the `Add` method.

```
PY
self.targets_collection.Add(portfolio_target)
```

To add a list of `PortfolioTarget` objects, call the `AddRange` method.

```
PY
self.targets_collection.AddRange(portfolio_targets)
```

Check Membership

To check if a `PortfolioTarget` exists in the `PortfolioTargetCollection` , call the `Contains` method.

```
target_in_collection = self.targets_collection.Contains(portfolio_target)
```

To check if a Symbol exists in the `PortfolioTargetCollection` , call the `ContainsKey` method.

```
symbol_in_collection = self.targets_collection.ContainsKey(symbol)
```

To get all the Symbol objects, use the `Keys` property.

```
symbols = self.targets_collection.Keys
```

Access Portfolio Targets

To access the `PortfolioTarget` objects for a Symbol, index the `PortfolioTargetCollection` with the Symbol.

```
portfolio_target = self.targets_collection[symbol]
```

To iterate through the `PortfolioTargetCollection` , call the `GetEnumerator` method.

```
enumerator = self.targets_collection.GetEnumerator()
```

To get all the `PortfolioTarget` objects, use the `Values` property

```
portfolio_targets = self.targets_collection.Values
```

Order Portfolio Targets by Margin Impact

To get an enumerable where position reducing orders are executed first and the remaining orders are executed in decreasing order value, call the `OrderByMarginImpact` method.

```
for target in self.targets_collection.OrderByMarginImpact(algorithm):
    # Place order
```

This method won't return targets for securities that have no data yet. This method also won't return targets for which the sum of the current holdings and open orders quantity equals the target quantity.

Remove Portfolio Targets

To remove a `PortfolioTarget` from the `PortfolioTargetCollection` , call the `Remove` method.

```
remove_successful = self.targets_collection.Remove(symbol)
```

To remove all the `PortfolioTarget` objects, call the `Clear` method.

```
self.targets_collection.Clear()
```

To remove all the `PortfolioTarget` objects that have been fulfilled, call the `ClearFulfilled` method.

```
self.targets_collection.ClearFulfilled(algorithm)
```

Copy Portfolio Targets

To copy a subset of the `PortfolioTarget` objects in the `PortfolioTargetCollection` to an array, call the `CopyTo` method. The `arrayIndex` argument is the zero-based index in the array at which copying begins.

```
self.targets_collection.CopyTo(array, arrayIndex)
```

Universe Timing Considerations

If the Execution model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Execution

Supported Models

Introduction

This page describes the pre-built Execution models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Execution model .

Null Model

The `NullExecutionModel` doesn't place any trades.

```
self.SetExecution(NullExecutionModel())
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Immediate Model

The `ImmediateExecutionModel` is the default Execution model. It uses market orders to immediately fill portfolio targets. It's similar to placing market orders in line with your algorithm logic.

```
self.SetExecution(ImmediateExecutionModel())
```

PY

To view the implementation of this model, see the LEAN GitHub repository .

Spread Model

The `SpreadExecutionModel` executes trades with market orders if the exchange is open and the bid-ask spread is within a threshold percentage. The model only works if the security subscription provides QuoteBar data. The spread percentage is calculated as

\frac{a - b}{p}

where a is the ask price, b is the bid price, and p is the price.

```
self.SetExecution(SpreadExecutionModel())
```

PY

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`acceptingSpreadPercent`	`float`	Maximum spread accepted comparing to current price in percentage	0.005 (0.5%)

To view the implementation of this model, see the LEAN GitHub repository .

Standard Deviation Model

The `StandardDeviationExecutionModel` seeks to fill orders when the price is more than 2 standard deviations lower than the average price over a trailing period. The intent is to find dips in the market to place trades. In strongly trending markets, this procedure can result in delayed trade placement since it might be a while before the next price spike or dip.

```
self.SetExecution(StandardDeviationExecutionModel())
```

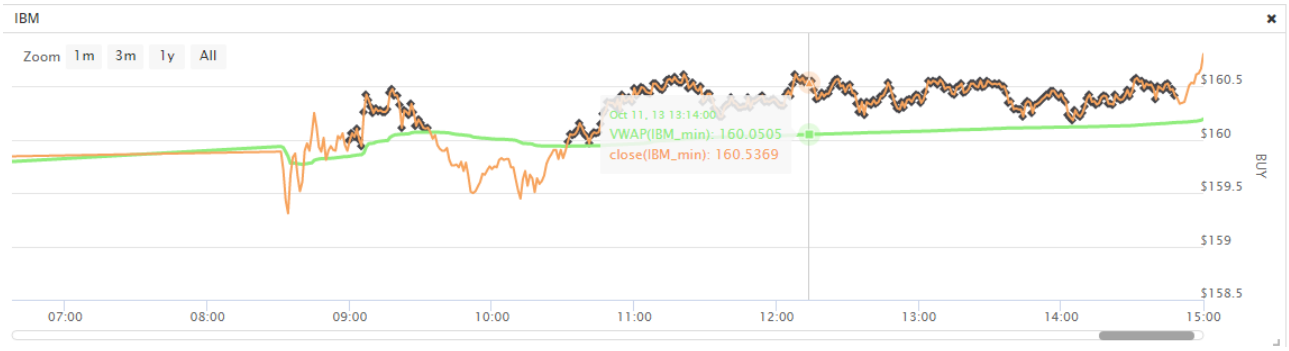
The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`period`	`float`	Period of the standard deviation indicator	60
`deviations`	`int`	The number of deviations away from the mean before submitting an order	2
`resolution`	`Resolution`	The resolution of the STD and SMA indicators	`Resolution.Minute`

To view the implementation of this model, see the LEAN GitHub repository .

VWAP Model

The `VolumeWeightedAveragePriceExecutionModel` works to fill your orders at or better than the volume-weighted average price (VWAP) for the trading day. This is a best-effort algorithm, so no guarantee can be made that it will reach the VWAP.



VWAP Execution Model Fill Placements

The model uses market orders and it only works if the security subscription provides intraday data. It sets a maximum order size of 1% of the current bar's volume, but you can update the maximum order size through the `MaximumOrderQuantityPercentVolume` member.

```
self.SetExecution(VolumeWeightedAveragePriceExecutionModel())
```

To view the implementation of this model, see the LEAN GitHub repository .

Algorithm Framework

Hybrid Algorithms

Introduction

Classic style algorithms can also use Algorithm Framework modules. This allows you to get the best of both styles of algorithm design, combining easily pluggable modules with the superior control of classic format.

Examples of popular use-cases of a hybrid approach are:

- Using framework universe selection models as the assets to select.
- Using portfolio construction models to decide portfolio allocations.
- Using a risk control model for free portfolio risk monitoring.
- Passing data between modules freely without concerns of the module interface.

Universe Selection

You can add one or more Framework Universe Selection Models to your algorithm and it will operate normally. You can also combine it with the `AddUniverse` method of the classic approach. The following example initializes a classic algorithm with framework universe selection models:

```
PY
def Initialize(self):
    self.SetUniverseSelection(ManualUniverseSelectionModel([ Symbol.Create("SPY", SecurityType.Equity,
Market.USA) ]))
    self.AddUniverseSelection(ManualUniverseSelectionModel([ Symbol.Create("AAPL", SecurityType.Equity,
Market.USA) ]))
    self.AddUniverse(self.CoarseSelection)
```

Alpha

You can add one or multiple Alpha models to your classic algorithm and place the orders using `Insight` objects without a Portfolio Construction model. To receive the collection of `Insight` objects in a your classic algorithm, implement the `InsightsGenerated` event handler:

```
PY
def Initialize(self):
    self.AddAlpha(EmaCrossAlphaModel())
    self.InsightsGenerated += self.OnInsightsGenerated

def OnInsightsGenerated(self, algorithm: IAlgorithm, insights_collection: GeneratedInsightsCollection) ->
None:
    insights = insights_collection.Insights
```

Portfolio Construction

You can add a Portfolio Construction model to your classic algorithm and have it place orders without returning `Insight` objects from an Alpha model. To emit insights without an Alpha model, in the `OnData` method, call the `EmitInsights` method.

The following example uses a Portfolio Construction Framework model with `EmitInsights` method in a classic algorithm:

```
PY
def Initialize(self):
    self.SetPortfolioConstruction(EqualWeightingPortfolioConstructionModel())

def OnData(self, slice):
 self.EmitInsights(Insight("GOOG", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up))
 self.EmitInsights([
  Insight("AAPL", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up),
  Insight("MSFT", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up)
 ])
```

Risk Management

Some Risk Management Models don't require a Portfolio Construction model to provide `PortfolioTarget` objects, allowing them to directly monitor the portfolio holdings and liquidate positions when neccessary. To see which pre-built Risk Management models don't need the Portfolio Construction model to provide `PortfolioTarget` objects, see Supported Models .

You can add one or more Risk Management Models to your algorithm and it will operate normally. The following example initializes a classic algorithm with framework risk management models:

```
PY
def Initialize(self):
    self.AddRiskManagement(MaximumDrawdownPercentPerSecurity(0.05))
    self.AddRiskManagement(MaximumUnrealizedProfitPercentPerSecurity(0.1))
```

Execution

Execution models can place orders for your strategy instead of placing them manually. To do so, PortfolioTarget s are routed to the `Execute` method of the Execution Model to place orders.

The following example uses a Framework Execution Model in a classic style algorithm:

```
PY
def Initialize(self):
    self.SetExecution(ImmediateExecutionModel())
```

Universe Timing Considerations

If the Alpha, Portfolio Construction, Risk Management, or Execution model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the

indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Algorithm Framework

Insight Manager

Introduction

The `InsightManager` tracks all of the Insight objects in your algorithm. It's an `InsightCollection` , which stores all of your insights in an internal dictionary that has the security `Symbol` as the key and a list of `Insight` objects as the value. You can access the manager anywhere in your algorithm where you have reference to the algorithm class. If you want to use an `InsightCollection` in your algorithm that's seperate from the `InsightManager` , create a new `InsightCollection` .

Add Insights

To add insights to the `InsightManager` , return a list of `Insight` objects from the Update method of your Alpha model or call the `EmitInsights` method. If you manage an `InsightCollection` that's seperate from the `InsightManager` , there are some methods to add `Insight` objects to it.

To add an insight to an `InsightCollection` , call the `Add` method.

```
self.insight_collection.Add(insight)
```

PY

To add a list of insights, call the `AddRange` method.

```
self.insight_collection.AddRange(insights)
```

PY

Check Membership

To check if an insight exists in the `InsightManager` , call the `Contains` method.

```
if self.algorithm.Insights.Contains(insight):
    pass
```

PY

To check if the `InsightManager` has an insight for a `Symbol` , call the `ContainsKey` method.

```
if self.algorithm.Insights.ContainsKey(symbol):
    pass
```

PY

To check if the `InsightManager` has active insights for a `Symbol` at a specific Coordinated Universal Time (UTC), call the `HasActiveInsights` method.

```
if self.algorithm.Insights.HasActiveInsights(symbol, utc_time):
    pass
```

Get Insights

To get the insights for a `Symbol` , index the `InsightManager` with the `Symbol` .

```
if self.algorithm.Insights.ContainsKey(symbol):
    insights = self.algorithm.Insights[symbol]
```

To get the insights that pass a filter, call the `GetInsights` method.

```
insights = self.algorithm.Insights.GetInsights(lambda insight: insight.Direction == InsightDirection.Up)
```

To iterate through the `InsightManager` , call the `GetEnumerator` method.

```
enumerator = self.algorithm.Insights.GetEnumerator()
```

To get all of the insights that will be active at a certain UTC time, call the `GetActiveInsights` method.

```
active_insights = self.algorithm.Insights.GetActiveInsights(utc_time)
```

Get the Next Expiry Time

To get the next insight expiry time in UTC, call the `GetNextExpiryTime` method.

```
next_expiry = self.algorithm.Insights.GetNextExpiryTime()
```

Remove Insights

Only the Portfolio Construction model should remove insights from the `InsightManager` . It should remove insights when the insights expire and when the corresponding security leaves the universe .

To remove an insight from the `InsightManager` , call the `Remove` method.

```
remove_successful = self.algorithm.Insights.Remove(insight)
```

To remove all the insights for a set of `Symbol` objects, pass a list of `Symbol` objects the `Clear` method.

```
self.algorithm.Insights.Clear(symbols)
```

To remove all the `Insight` objects, call the `Clear` method.

```
PY
self.algorithm.Insights.Clear()
```

To remove all the insights that will be expired at a certain UTC time, call the `RemoveExpiredInsights` method.

```
PY
expired_insights = self.algorithm.Insights.RemoveExpiredInsights(utc_time)
```

Cancel Insights

In some cases, you may want to cancel an Insight. For example, if a Risk Management model in your algorithm liquidates a security, it should also cancel all of the active insights for the security. If you don't cancel the insights, the Portfolio Construction model might create a new PortfolioTarget to re-enter the position.

Another example of a situtation where you would want to cancel an Insight is when an Alpha model in your algorithm determines the trading opportunity has pre-maturely ended. For instance, say you want your algorithm to enter a long position in a security when its Relative Strength Index (RSI) moves below 20 and then liquidate the position when the security's RSI moves above 30. If you emit an Insight that has a duration of 30 days when the RSI moves below 20 but the RSI moves above 30 in 10 days, you should cancel the Insight when the RSI moves above 30.

To cancel insights, call the `Cancel` / `Expire` method with a list of `Insight` objects.

```
PY
algorithm.Insights.Cancel(insights)
```

To cancel all the insights for some securities, call the `Cancel` / `Expire` method with a list of `Symbol` objects.

```
PY
algorithm.Insights.Cancel(symbols)
```

When you cancel an active insight, it's `CloseTimeUtc` property is set to one second into the past.

Copy Insights

To copy a subset of the `Insight` objects in the `InsightManager` to an array, call the `CopyTo` method. The `arrayIndex` argument is the zero-based index in the array at which copying begins.

```
PY
self.algorithm.Insights.CopyTo(array, arrayIndex)
```

Preserve Insights Between Deployments

Follow these steps to use the Object Store to preserve the algorithm state across live deployments:

1. Create an algorithm that defines a storage key and adds insights to the Insight Manager .

```
class ObjectStoreChartingAlgorithm(QCAlgorithm):
    def Initialize(self):
        self.insight_key = f"{self.ProjectId}/insights"
        self.SetUniverseSelection(ManualUniverseSelectionModel([ Symbol.Create("SPY",
SecurityType.Equity, Market.USA) ]))
        self.SetAlpha(ConstantAlphaModel(InsightType.Price, InsightDirection.Up, timedelta(5), 0.025,
None))
```

2. At the top of the algorithm file, add the following imports:

```
from Newtonsoft.Json import JsonConvert
from System.Collections.Generic import List
```

`Insight` objects are a C# objects, so you need the preceding C# libraries to serialize and deserialize them.

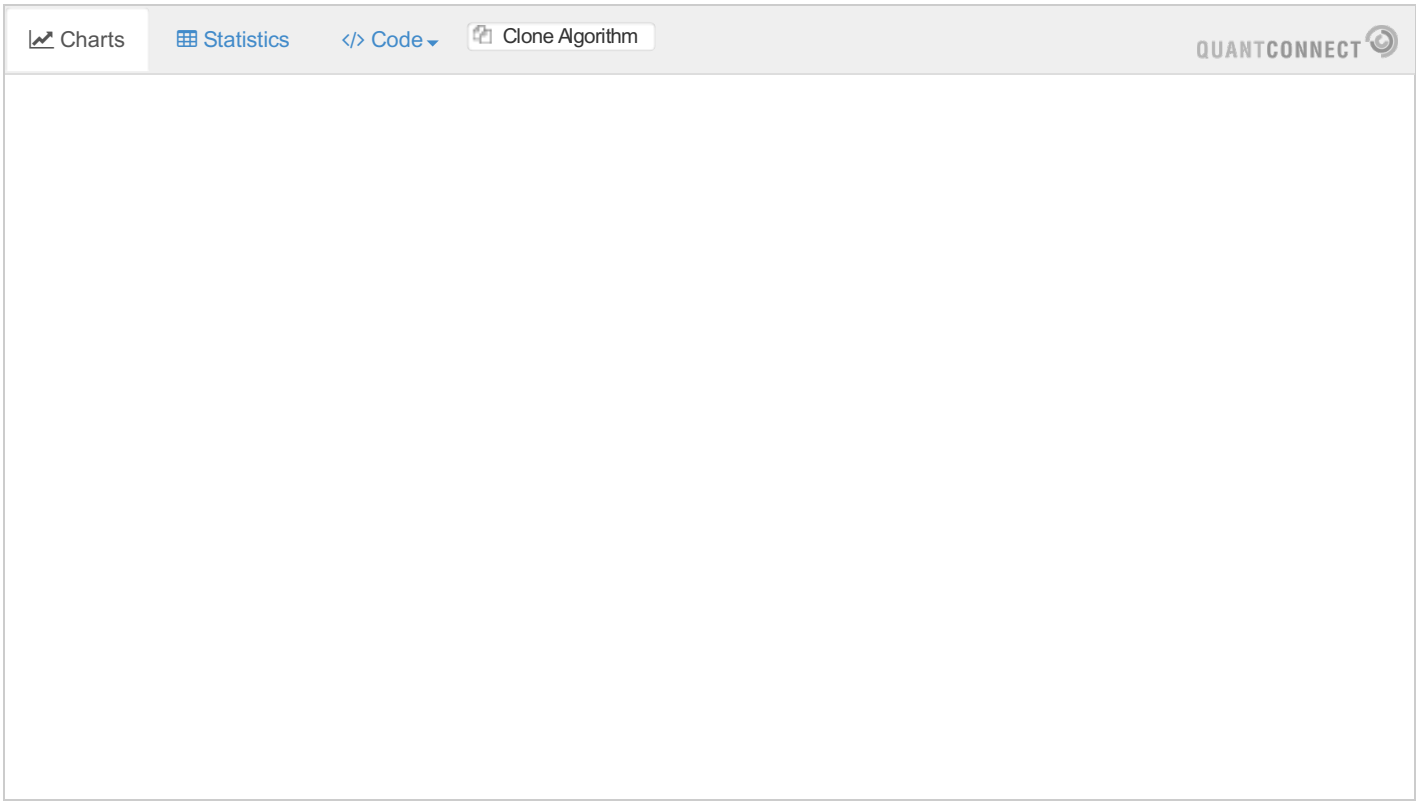
3. In the OnEndOfAlgorithm event handler of the algorithm, get the Insight objects and save them in the Object Store as a JSON object.

```
def OnEndOfAlgorithm(self):
    insights = self.Insights.GetInsights(lambda x: x.IsActive(self.UtcTime))
    content = ','.join([JsonConvert.SerializeObject(x) for x in insights])
    self.ObjectStore.Save(self.insight_key, f'[{content}]')
```

4. At the bottom of the `Initialize` method, read the Insight objects from the Object Store and add them to the Insight Manager .

```
if self.ObjectStore.ContainsKey(self.insight_key):
    insights = self.ObjectStore.ReadJson[List[Insight]](self.insight_key)
    self.Insights.AddRange(insights)
```

The following algorithm provides a full example of preserving the Insight state between deployments:



Properties

The `InsightManager` has the following properties:

Charting

Introduction

We provide a powerful charting API that you can use to build many chart types.

Charts

Charts contain a collection of series, which display data on the chart. To add a chart to an algorithm, create a `Chart` object and then call the `AddChart` method.

```py
chart = Chart("<chartName>")
self.AddChart(chart)
```

The `Chart` constructor expects a name argument. The following chart names are reserved:

- Assets Sales Volume
- Exposure
- Alpha Assets
- Alpha Asset Breakdown

Series

A chart series displays data on the chart. To add a series to a chart, create a `Series` object and then call the `AddSeries` method.

```py
series = Series("<seriesName>")
chart.AddSeries(series)
```

Arguments

There are several other headers for the `Series` constructor.

```
Series(name, type)
Series(name, type, index)
Series(name, type, index, unit)
Series(name, type, unit)
Series(name, type, unit, color)
Series(name, type, unit, color, symbol)
```

The following table describes the constructor arguments:

Argument	Data Type	Description
`name`	`str`	Name of the series
`type`	`SeriesType`	Type of the series
`index`	`int`	Index position on the chart of the series
`unit`	`str`	Unit for the series axis
`color`	`Color`	Color of the series
`symbol`	`ScatterMarkerSymbol`	Symbol for the marker in a scatter plot series

The default `Series` is a line chart with a "$" unit on index 0.

Names

The `Series` constructor expects a name argument. If you add a series to one of the default charts, some series names may be reserved. The following table shows the reserved series name for the default charts:

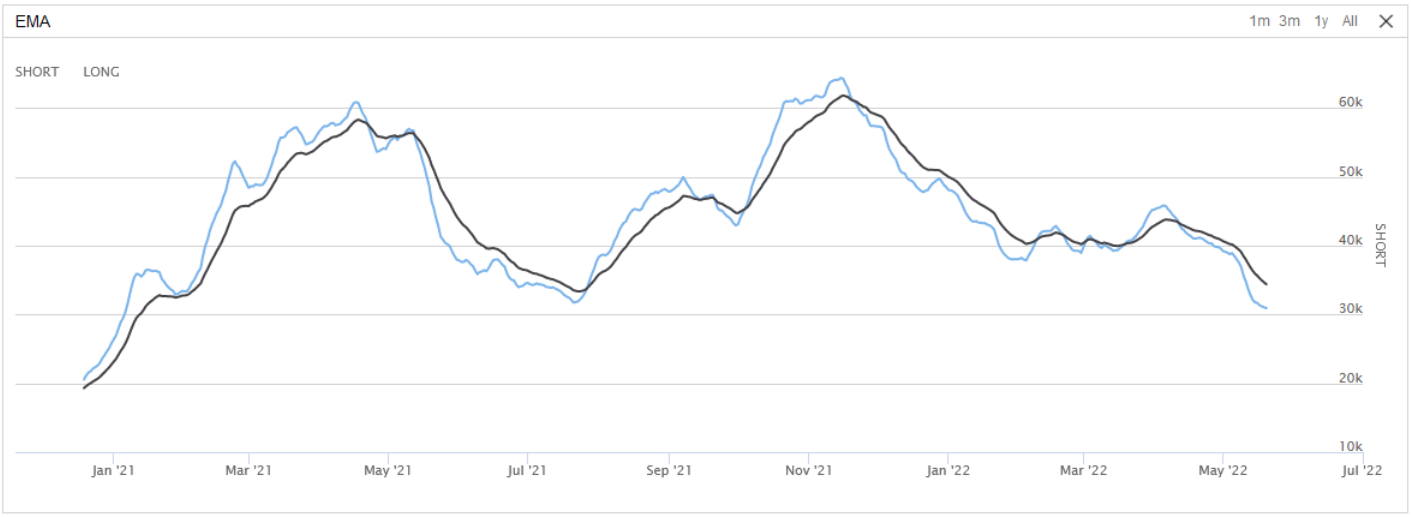
Chart Name	Reserved Series Names
Strategy Equity	Equity, Daily Performance
Capacity	Strategy Capacity
Drawdown	Equity Drawdown
Benchmark	Benchmark

Types

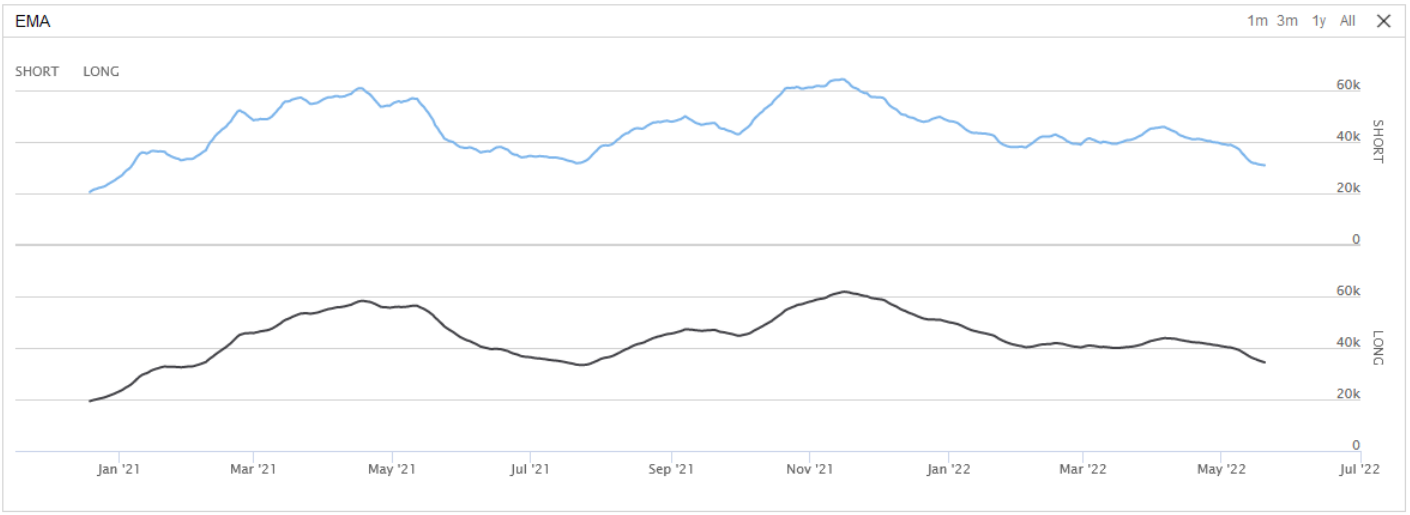
The `SeriesType` enumeration has the following members:

Index

The series index refers to its position in the chart. If all the series are at index 0, they lay on top of each other. If each series has its own index, each series will be separate on the chart. The following image shows an EMA cross chart with both EMA series set to the same index:



The following image shows the same EMA series, but with the short EMA on index 0 and the long EMA on index 1:



Colors

To view the available `Color` options, see the Color Struct Properties in the .NET documentation.

Scatter Marker Symbols

The `ScatterMarkerSymbol` enumeration has the following members:

Plot Data

To add a data point to a chart series, call the `Plot` method. If you haven't already created a chart and series with the names you pass to the `Plot` method, the chart and/or series is automatically created.

```
self.Plot("<chartName>", "<seriesName&gt", value)
```

The `value` argument can be an integer for decimal number. If the chart is a time series, the value is added to the chart using the algorithm time as the x-coordinate.

To plot the current value of indicators, call the `Plot` method. The method accepts up to four indicators.

```py
# In Initialize
symbol = self.AddEquity("SPY")
sma_short = self.SMA(symbol, 10)
sma_long = self.SMA(symbol, 20)

# In OnData
self.Plot("<chartName>", sma_short, sma_long)
```

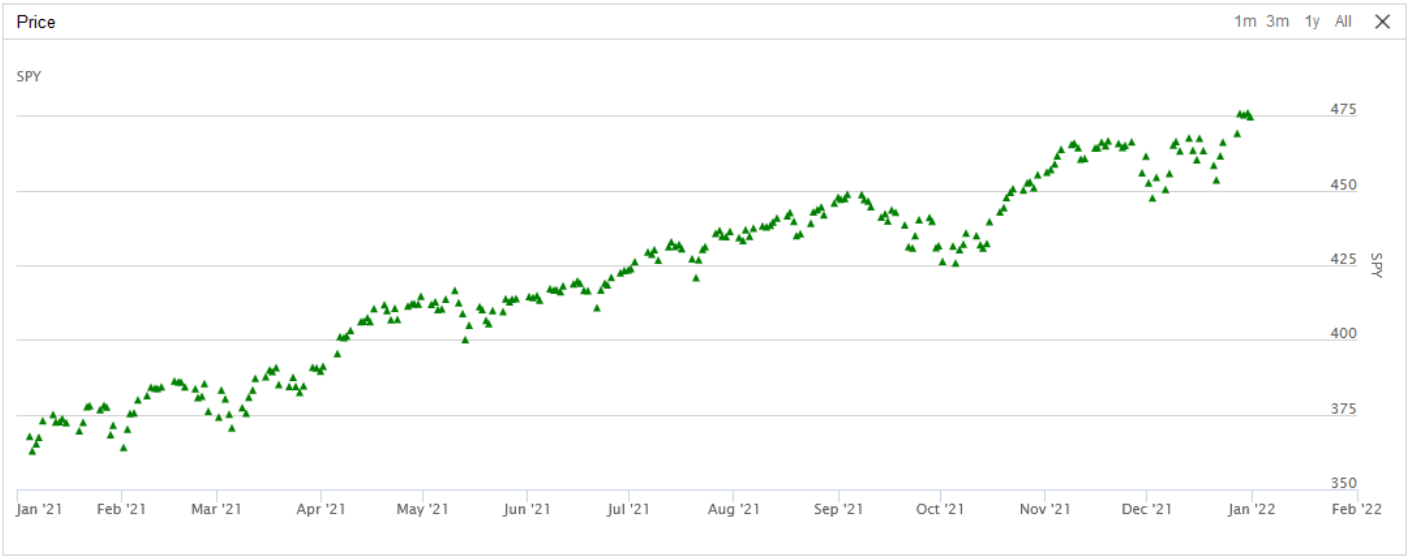
To plot all of the values of some indicators, in the `Initialize` method, call the `PlotIndicator` method. The method plots each indicator value as the indicator updates. The method accepts up to four indicators.

```py
symbol = self.AddEquity("SPY")
sma_short = self.SMA(symbol, 10)
sma_long = self.SMA(symbol, 20)
self.PlotIndicator("<chartName>", sma_short, sma_long)
```

Examples

The following example shows how to plot the daily closing price of SPY with a scatter plot:

```py
class ChartingDemoAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2021, 1, 1)
        self.SetEndDate(2022, 1, 1)
        self.AddEquity("SPY", Resolution.Daily)
        chart = Chart("Price")
        self.AddChart(chart)
        chart.AddSeries(Series("SPY", SeriesType.Scatter, "$", Color.Green, ScatterMarkerSymbol.Triangle))

    def OnEndOfDay(self, symbol: Symbol) -> None:
        self.Plot("Price", "SPY", self.Securities[symbol].Price)
```



To see a full example, run the CustomChartingAlgorithm .

View Charts

The following table describes where you can access your charts, depending on how to deploy your algorithms:

Location	Algorithm Lab Algorithms	CLI Cloud Algorithms	CLI Local Algorithms
Backtest results page	✓	✓	
Live results page	✓	✓	
/backtests/read endpoint	✓	✓	
/live/read endpoint	✓	✓	
ReadBacktest method	✓	✓	
ReadLiveAlgorithm method	✓	✓	
Local **JSON** file in your **<projectName> / backtests / <timestamp>** or **<projectName> / live / <timestamp>** directory		✓	✓

Quotas

Custom charts are limited to 4,000 data points. Intensive charting requires hundreds of megabytes of data, which is too much to stream online or display in a web browser. If you exceed the quota, the following message displays:

Exceeded maximum points per chart, data skipped

You can create up to 10 custom chart series per algorithm. If you exceed the quota, your algorithm stops executing and the following message displays:

Exceeded maximum series count: Each backtest can have up to 10 series in total.

In live trading, charts are sampled every one and ten minutes. If you create 1-minute resolution custom charts, the IDE charting will downgrade the granularity and display the 10-minutes sampling after a certain amount of samples.

Logging

Introduction

Algorithms can record string messages ('log statements') to a file for analysis after a backtest is complete, or as a live algorithm is running. These records can assist in debugging logical flow errors in the project code. Consider adding them in the code block of an `if` statement to signify an error has been caught.

It's good practice to add logging statements to live algorithms so you can understand its behavior and keep records to compare against backtest results. If you don't add logging statements to a live algorithm and the algorithm doesn't trade as you expect, it's difficult to evaluate the underlying problem.

Log Messages

Log statements are added to the log file while your algorithm continues executing. Logging dataset information is not permitted. Use `Log` statements to debug your backtests and live trading algorithms.

If you execute algorithms in QuantConnect Cloud, log length is capped by organization tier . If your organization hits the daily limit, contact us .

If you log the same content multiple times, only the first instance is added to the log file. To bypass this rate-limit, add a timestamp to your log messages.

For live trading, the log files of each cloud project can store up to 100,000 lines for up to one year. If you log more than 100,000 lines or some lines become older than one year, we remove the oldest lines in the files so your project stays within the quota.

To record the algorithm state when the algorithm stops executing, add log statements to the `OnEndOfAlgorithm` event handler.

```
PY
self.Log("My log message")
```

Debug Messages

Debug statements are the same as log statements, but `Debug` statements are orange in the Cloud Terminal. Use these statements when you want to give more attention to a message in the Cloud Terminal. Debug messages can be up to 200 characters in length. If you send multiple debug statements within 1 second, your messages are rate-limited to avoid crashing your browser.

```
PY
self.Debug("My debug message")
```

Error Messages

Error statements are the same as log statements, but `Error` statements are displayed in red text in the Cloud Terminal.

Use these statements when you want to give the most attention to a message in the Cloud Terminal. Error statements are rate-limited like debug statements.

```
self.Error("My error message")
```

Quit Messages

Quit statements cause your project to stop running and may log some data to the log file and Cloud Terminal. These statements are orange in the Cloud Terminal. When you call the `Quit` method, the program continues executing until the end of the method definition. If you want to quit execution immediately, `return` after you call `Quit` .

```
self.Quit("My quit message")
```

Runtime Statistics

Runtime statistics show the performace of your algorithm at a single moment in time.

The following table describes the default runtime statistics:

Statistic	Description
Capacity	The maximum amount of money an algorithm can trade before its performance degrades from market impact.
Equity	The total portfolio value if all of the holdings were sold at current market rates.
Fees	The total quantity of fees paid for all the transactions.
Holdings	The absolute sum of the items in the portfolio.
Net Profit	The dollar-value return across the entire trading period.
PSR	The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).
Return	The rate of return across the entire trading period.
Unrealized	The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.
Volume	The total value of assets traded for all of an algorithm's transactions.

The capacity statistic is only available for backtests.

To add a custom runtime statistic, call the `SetRuntimeStatistic` method with a `name` and `value` . The `value` argument can be a `string` or a number.

```
self.SetRuntimeStatistic(name, value)
```

Access Logs and Statistics

The following tables describe how to access the logs and runtime statistics, depending on how to deploy your algorithms.

QuantConnect Cloud Deployments

The following table describes how to access the logs and runtime statistics of algorithms you deploy in QuantConnect Cloud:

Execution Type	Result Type	Access Tools
Backtest	Logs	• Backtest results page in the Algorithm Lab • Backtest results page on Local Platform
Backtest	Runtime Statistics	• Backtest results page in the Algorithm Lab • Backtest results page on Local Platform • ReadBacktest API method • /backtests/read endpoint
Live Trading	Logs	• Live results page in the Algorithm Lab • Live results page on Local Platform • /live/read/log endpoint
Live Trading	Runtime Statistics	• Live results page in the Algorithm Lab • Live results page on Local Platform • ReadLiveAlgorithm API method • /live/read endpoint

On-Premise Deployments

The following table describes how to access the logs and runtime statistics of algorithms you deploy on-premise:

Execution Type	Result Type	Access Tools
Backtest	Logs	• Backtest results page on Local Platform • **<organizationWorkspace> / <projectName> / backtests / <unixTimestamp> / <algorithmId>-log.txt** • lean logs CLI command
Backtest	Runtime Statistics	• Backtest results page on Local Platform • **<organizationWorkspace> / <projectName> / backtests / <unixTimestamp> / <algorithmId>.json**
Live Trading	Logs	• Live results page on Local Platform • **<organizationWorkspace> / <projectName> / live / <timeStamp> / / L-<deploymentId>-log.txt** • lean logs CLI command
Live Trading	Runtime Statistics	• Live results page on Local Platform • **<organizationWorkspace> / <projectName> / live / <timeStamp> / / L-<deploymentId>.json**

Live Trading

QuantConnect enables you to run your algorithms in live mode with real-time market data. You can use the same algorithm for backtesting and live trading with different brokerages and data feeds.

Key Concepts

Synchonize your account and algorithm

Brokerages

Stream data into your algorithm

Data Feeds

Send orders to your brokerage

Trading and Orders

Reconciliation

Differences from backtesting

Notifications

Stay informed on algorithm decisions

Charting and Logging

Display information on algorithm decisions

Signal Exports

Send portfolio targets to third-party services

See Also

QuantConnect Live Trading
Supported Brokerages

Live Trading

Key Concepts

Introduction

Algorithms should generally work seamlessly shifting from backtesting to live trading with no changes required. In backtesting algorithm events are triggered as fast as possible, whereas in live trading events are triggered in realtime.

Algorithm Portfolio

The live trading algorithm portfolio is loaded from the brokerage holdings, including open orders. When there are unsupported asset types in the portfolio the live trading strategy is stopped as we cannot be sure how to model the asset.

In paper trading the algorithm starts from no portfolio, and attempts to remember the previous portfolio state from the last time the portfolio was deployed.

Your algorithm should not assume it starts from an empty portfolio as you may have holdings from previous deployments.

Stateful Strategies

There is no automatic state management of live strategies in QuantConnect. We recommend algorithms reconstruct state or indicator needs using the WarmUp and History methods.

Once the algorithm has warmed up some objects you can use the Object Store to save the data for the next algorithm restart.

In the event the algorithm is terminated unexpectedly, you should review the live algorithm portfolio at the brokerage to confirm it will behave as expected.

Data Delivery Times

Most data is available at the same time as in backtesting, with the exception of daily data which is often delayed by 4-8 hours (4am - 8am).

US Equities universe selection data is processed at 7-8am ET.

Other Brokerages

If we do not support your brokerage reach out to let us and we can attempt to add them to the roadmap. With enough community requests we can build and open-source the brokerage integration.

Live Trading

Brokerages

Introduction

Brokerages provide you with a connection to the market so you can fill trades. To avoid placing invalid orders for execution in live trading, LEAN validates your orders before sending them to the real brokerage. To view all of the integrated brokerages, see Brokerages .

Portfolio

In live trading, LEAN populates the `Portfolio` object with your account holdings and the `Transactions` object with your open positions. If you don't manually subscribe to the assets in your account, LEAN subscribes to them with the lowest resolution of the subscriptions in your algorithm. For example, say you hold AAPL shares in your account and create the following subscriptions in your algorithm:

```
self.AddEquity("SPY", Resolution.Hour)
self.AddEquity("MSFT", Resolution.Second)
```

In this case, LEAN subscribes to second-resolution data for AAPL since the lowest resolution in your algorithm is `Resolution.Second` .

Deposits and Withdraws

You can deposit and withdraw cash from your brokerage account while you run an algorithm that's connected to the account. We sync the algorithm's cash holdings with the cash holdings in your brokerage account every day at 7:45 AM Eastern Time (ET).

Monitor the Brokerage Connection

We notify your algorithm when your brokerage connection disconnects and reconnects.

Lost Connections

If the brokerage connection breaks, we notify your algorithm through the `OnBrokerageDisconnect` event handler.

```
def OnBrokerageDisconnect(self) -> None:
    self.Debug("Brokerage connection lost")
```

Restored Connections

When the brokerage connection restores after it disconnects, we notify your algorithm through the `OnBrokerageReconnect` event handler.

```
def OnBrokerageReconnect(self) -> None:
    self.Debug("Brokerage connection restored")
```

Example Algorithm

For a full example algorithm that implements the `OnBrokerageDisconnect` and `OnBrokerageReconnect` methods, see the BrokerageActivityEventHandlingAlgorithm in the LEAN GitHub repository.

Monitor Brokerage Messages

When your brokerage sends you a message, we notify your algorithm through the `OnBrokerageMessage` event handler.

```
def OnBrokerageMessage(self, message_event: BrokerageMessageEvent) -> None:
    self.Debug(f"Brokerage message received: {message_event.Message}")
```

`BrokerageMessageEvent` objects have the following attributes:

For a full example algorithm that implements the `OnBrokerageMessage` method, see the BrokerageActivityEventHandlingAlgorithm in the LEAN GitHub repository.

To handle brokerage messages outside of your algorithm class, create and set a `BrokerageMessageHandler` . For a full example, see the CustomBrokerageMessageHandlerAlgorithm in the LEAN GitHub repository.

Live Trading

Data Feeds

Introduction

Data feeds are a stream of asset prices and quotes delivered to your trading algorithm during live execution. You need live data feeds to inject data into your algorithm so that you can make real-time trading decisions and so that the values of the securities in your portfolio update in real-time. You can source data from QuantConnect or your brokerage. To view the available data feeds, see Data Feeds .

Bar Building

We aggregate ticks to build bars.

In live trading, bars are built using the exchange timestamps with microsecond accuracy. This microsecond-by-microsecond processing of the ticks can mean that the individual bars between live trading and backtesting can have slightly different ticks. As a result, it's possible for a tick to be counted in different bars between backtesting and live trading, which can lead to bars having slightly different open, high, low, close, and volume values.

Latency

Live data takes time to travel from the source to your algorithm. The data feed has a latency of 20-50 milliseconds. QuantConnect is not designed for high-frequency trading.

History Requests

In live trading, if you make a history request for minute data at noon and the history period covers the start of the previous day to the present moment, the data from the previous day will be backtest data. The data of the current day will be live data that we collected throughout the morning. If you make this history request in a backtest, you might get slightly different data for the current day because of post-processing from the data vendor.

Warm Up Periods

LEAN supports an automated fast-forward system called "Warm Up". It simulates winding back the clock from the time you deploy the algorithm. In a backest, this is the `StartDate` of your algorithm. In live trading, it's the current date. Warm Up is a great way to prepare your algorithm and its indicators for trading.

US Fundamental Data by Morningstar is limited to the last 30 days.

Live Trading

Trading and Orders

Introduction

LEAN has dozens of methods to create, update, and cancel orders. You can place orders automatically with helper methods or manually through methods on the algorithm API. You can fetch, update, and cancel manual orders with order tickets . As the state of your orders change, LEAN creates events that notify your algorithm.

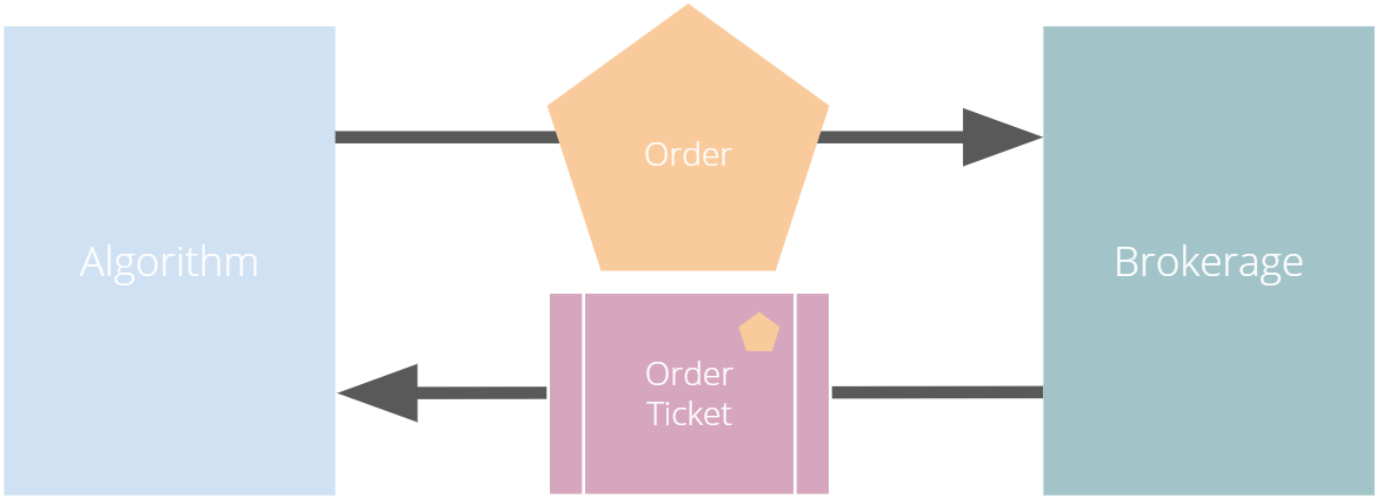
In backtesting, LEAN simulates order fills with historical data, but you can create your own fill, fee, slippage, and margin models via plugin points. You control how optimistic or pessimistic order fills are with transaction model classes. For more information about these types of models, see Reality Modeling .

In live trading, orders fill asynchronously. We send your order to the API of your brokerage and wait for their response to update the state of your algorithm and portfolio. The timing of live order fills doesn't depend on the resolution of your security subscriptions. When your order fills, the fill price and fee is set by your brokerage. You can add event handlers to your algorithm to monitor the brokerage connection and brokerage messages .

In backtesting, the trade fill timing depends on the resolution of your security subscription. For example, if you subscribe to a security with minute resolution data, the algorithm only receives data in one-minute time slices . As a result, the fill model can only evaluate if the order should fill on a minute-by-minute frequency.

Order Life Cycle

When you call one of the order methods to place an order, LEAN performs pre-order checks to ensure that the order meets some requirements and uses the last known price to check you have enough capital . To see the requirements of each order, see the **Requirements** section of the documentation for the order types and the **Orders** section of your brokerage model . If you can place the order, LEAN creates `Order` and OrderTicket objects. The order ticket is sent to your brokerage. As the brokerage processes your order, it returns another order ticket and it's compared against the order to see if the order is satisfied. Orders are asynchronous in live trading, so if you want to change an order, you must request it with the order ticket. Order changes are not guaranteed since your order may fill by the time brokerage receives the request.



When you create an order ticket, LEAN validates it to avoid sending orders that will be rejected by your brokerage. If the order is invalid, it's status is set to `OrderStatus.Invalid` . Otherwise, LEAN sets the order status to `OrderStatus.New` , sends the order to your brokerage, and waits for the brokerage message. If your brokerage accepts the order, the order status becomes `OrderStatus.Submitted` . Otherwise, it becomes `OrderStatus.Canceled` .

You can update orders until they fill or the brokerage prevents modifications. If you place an order update request, LEAN validates it to avoid sending orders that will be rejected by you brokerage. If the order update is valid, LEAN sends the update request and waits for the brokerage message. If your brokerage accepts the update, the order status becomes `OrderStatus.UpdateSubmitted` . Otherwise, it becomes `OrderStatus.Canceled` .

The brokerage notifies LEAN when you order fills (`OrderStatus.Filled`), partially fills (`OrderStatus.PartiallyFilled`) or is canceled (`OrderStatus.Canceled`). The brokerage can cancel your order before it completely fills. For example, if you place a market on open and the asset price moves before the market open to where you can't afford the quantity of the order, the brokerage rejects your order. The brokerage can also cancel your market on open order after partial fills if there is no shares left to trade.

Disable Buying Power

You can disable order margin checks and opt to let your brokerage decide to accept or reject the trades. This is helpful in live trading if you have a more permissive brokerage margin allowance that what LEAN models. The default position group buying power models are helpful for Option trading strategies. However, it can be counterproductive if it's not a supported Option strategy . To disable the validations of the default position group buying power model, use the `NullSecurityPositionGroupModel` . To set the `NullSecurityPositionGroupModel` for the portfolio, during initialization , call the `SetPositions` method with the `SecurityPositionGroupModel.Null` argument.

```
self.Portfolio.SetPositions(SecurityPositionGroupModel.Null)
```

PY

To disable the validations of the default buying power model , use the `NullBuyingPowerModel` . To set the `NullBuyingPowerModel` for a security subscription, call the `SetBuyingPowerModel` method with the `BuyingPowerModel.Null` argument.

```
equity = self.AddEquity("SPY")
equity.SetBuyingPowerModel(BuyingPowerModel.Null)
# Alias:
# equity.SetMarginModel(SecurityMarginModel.Null)
```

You can also set the asset `NullBuyingPowerModel` in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to set the buying power of securities in the security initializer, call `SetSecurityInitializer` before you create security subscriptions and after you call `SetBrokerageModel` .

```
# In Initialize
self.SetSecurityInitializer(MySecurityInitializer(self.BrokerageModel,
FuncSecuritySeeder(self.GetLastKnownPrices)))

# Outside of the algorithm class
class MySecurityInitializer(BrokerageModelSecurityInitializer):

    def __init__(self, brokerage_model: IBrokerageModel, security_seeder: ISecuritySeeder) -> None:
        super().__init__(brokerage_model, security_seeder)

    def Initialize(self, security: Security) -> None:
        # First, call the superclass definition
        # This method sets the reality models of each security using the default reality models of the
brokerage model
        super().Initialize(security)

        # Next, overwrite the security buying power
        security.SetBuyingPowerModel(BuyingPowerModel.Null)
```

Trading and Orders

Financial Advisors

Introduction

Financial Advisor accounts enable certified professionals to use a single trading algorithm to manage several client accounts. Our Interactive Brokers integration enables you to place FA group orders.

Group Routing

To place trades using a subset of client accounts, create Account Groups in Trader Workstation and then define the `InteractiveBrokersOrderProperties` when you create orders.

```py
self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
self.DefaultOrderProperties.FaGroup = "TestGroupEQ"
self.DefaultOrderProperties.FaMethod = "EqualQuantity"
self.DefaultOrderProperties.FaProfile = "TestProfileP"
self.DefaultOrderProperties.Account = "DU123456"
```

SecurityHolding objects aggregate your positions across all the account groups. If you have two groups where group A has 10 shares of SPY and group B has -10 shares of SPY, then `self.Portfolio["SPY"].Quantity` is zero.

Allocation Methods

The following table shows the supported allocation methods for FA group orders:

`FaMethod`	**Description**
`"EqualQuantity"`	Distributes shares equally between all accounts in the group. If you use this method, specify an order quantity.
`"NetLiq"`	Distributes shares based on the net liquidation value of each account. The system calculates ratios based on the net liquidation value in each account and allocates shares based on these ratios. If you use this method, specify an order quantity.
`"AvailableEquity"`	Distributes shares based on the amount of available equity in each account. The system calculates ratios based on the available equity in each account and allocates shares based on these ratios. If you use this method, specify an order quantity.
`"PctChange"`	Increases or decreases an already existing position. Positive percents increase positions and negative percents decrease positions. If you use this method, specify a percent instead of an order quantity.

```
def Initialize(self) -> None:
    # Set the default order properties
    self.DefaultOrderProperties = InteractiveBrokersOrderProperties()
    self.DefaultOrderProperties.FaGroup = "TestGroupEQ"
    self.DefaultOrderProperties.FaMethod = "EqualQuantity"
    self.DefaultOrderProperties.FaProfile = "TestProfileP"
    self.DefaultOrderProperties.Account = "DU123456"

def OnData(self, slice: Slice) -> None:
    # Override the default order properties
    # "NetLiq" requires a order size input
    order_properties = InteractiveBrokersOrderProperties()
    order_properties.FaMethod = "NetLiq"
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    # "AvailableEquity" requires a order size input
    order_properties.FaMethod = "AvailableEquity"
    self.LimitOrder(self.symbol, quantity, limit_price, orderProperties=order_properties)

    # "PctChange" requires a percentage of portfolio input
    order_properties.FaMethod = "PctChange"
    self.SetHoldings(self.symbol, pct_portfolio, orderProperties=order_properties)
```

PY

Subscription Requirements

To use FA group orders through our Interactive Brokers integration, you need to connect as a member of a Trading Firm and Institution organization. If you aren't currently on either of these tiers, upgrade your organization .

Live Trading

Reconciliation

Introduction

Algorithms usually perform differently between backtesting and live trading over the same time period. Backtests are simulations where we model reality as close as possible, but the modeling isn't always perfect. To measure the performance differences, we run an out-of-sample (OSS) backtest in parallel to all of your live trading deployments. The live results page displays the live equity curve and the OOS backtest equity curve of your algorithms.

If your algorithm is perfectly reconciled, it has an exact overlap between its live and OOS backtest equity curves. Deviations mean that the performance of your algorithm has differed between the two execution modes. Several factors can contribute to the deviations.

Differences From Data

The data that your algorithm uses can cause differences between backtesting and live trading performance.

Look-Ahead Bias

The Time Frontier minimizes the risk of look-ahead bias in backtests, but it does not completely eliminate the risk of look-ahead bias. For instance, if you use a custom dataset that contains look-ahead bias, your algorithm's live and backtest equity curves may deviate. To avoid look-ahead bias with custom datasets, set a `Period` on your custom data points so that your algorithm receives the data points after the `Time` + `Period` .

Discrete Time Steps

In backtests, we inject data into your algorithm at predictable times, according to the data resolution. In live trading, we inject data into your algorithm when new data is available. Therefore, if your algorithm has a condition with a specific time (i.e. time is 9:30:15), the condition may work in backtests but it will always fail in live trading since live data has microsecond precision. To avoid issues, either use a time range in your condition (i.e. 9:30:10 < time < 9:30:20), use a rounded time, or use a Scheduled Event.

Custom Data Emission Times

Custom data is often timestamped to midnight, but the data point may not be available in reality until several days after that point. If your custom dataset is prone to this delay, your backtest may not fetch the same data at the same time or frequency that your live trading algorithm receives the data, leading to deviations between backtesting and live trading. To avoid issues, ensure the timestamps of your custom dataset are the times when the data points would be available in reality.

In backtesting, LEAN and custom data are perfectly synchonized. In live trading, daily and hourly data from a custom data source are not because of the frequency that LEAN checks the data source depends on the `resolution` argument. The following table shows the polling frequency of each resolution:

Resolution	Update Frequency
Daily	Every 30 minutes
Hour	Every 30 minutes
Minute	Every minute
Second	Every second
Tick	Constantly checks for new data

Split Adjustment of Indicators

Backtests use adjusted price data by default. Therefore, if you don't change the data normalization mode , the indicators in your backtests are updated with adjusted price data. In contrast, if a split or dividend occurs in live trading, your indicators will temporarily contain price data from before the corporate event and price data from after the corporate event. If this occurs, your indicators will produce different signals in your backtests compared to your live trading deployment. To avoid issues, reset and warm up your indicators when your algorithm receives a corporate event.

Tick Slice Sizes

In backtesting, we collect ticks into slices that span 1 millisecond before injecting them into your algorithm. In live trading, we collect ticks into slices that span up to 70 milliseconds before injecting them into your algorithm. This difference in slice sizes can cause deviations between your algorithm's live and OOS backtest equity curves. To avoid issues, ensure your strategy logic is compatible with both slice sizes.

Differences From Modeling

The modeling that your algorithm uses can cause differences between backtesting and live trading performance.

Reality Modeling Error

We provide brokerage models to model fees, slippage, and order fills in backtests. However, these model predictions may not always match the fees that your live algorithm incurs, leading to deviations between backtesting and live trading. You can adjust the reality models that your algorithm uses to more accurately reflect the specific assets that you're trading. For more information about reality models, see Reality Modeling .

Market Impact

We don't currently model market impact. So, if you are trading large orders, your fill prices can be better during backtesting than live trading, causing deviations between backtesting and live trading. To avoid issues, implement a custom fill model in your backtests that incorporates market impact.

Fills

In backtests, orders fill immediately. In live trading, they are sent to your brokerage and take about half a second to execute. If you fill an order in a backtest with stale data, deviations between backtesting and live trading can occur because the order is filled at a price that is likely different from the real market price. Stale order fills commonly occur in backtests when you create a Scheduled Event with an incompatible data resolution. For instance, if you subscribe to hourly data, place a Scheduled Event for 11:15 AM, and fill an order during the Scheduled Event, the order will fill at a stale price because the data between 11:00 AM and 11:15 AM is missing. To avoid stale fills, only place orders when your algorithm receives price data.

In live trading, your brokerage provides the fill price of your orders. Since the backtesting brokerage models do not know the price at which live orders are filled, the fill price of backtest orders is based on the best price available in the current backtesting data. Similarly, limit orders can fill at different prices between backtesting and live trading. In backtesting, limit orders fill as soon as the limit price is hit. In live trading, your brokerage may fill the same limit order at a different price or fail to fill the order, depending on the position of your order in their order book.

Borrowing Costs

We do not currently simulate the cost of borrowing shorts in backtests. Therefore, if your algorithm takes short positions, deviations can occur between backtesting and live trading. We are working on adding the functionality to model borrowing fees. Subscribe to GitHub Issue #4563 to track the feature progress.

Differences From Brokerage

The brokerage that your algorithm uses can cause differences between backtesting and live trading performance.

Portfolio Allocations on Small Accounts

If you trade a small portfolio, it's difficult to achieve accurate portfolio allocations because shares are usually sold in whole numbers. For instance, you likely can't allocate exactly 10% of your portfolio to a security. You can use fractional shares to achieve accurate portfolio allocations, but not all brokerages support fractional shares. To get the closest results when backtesting and live trading over the same period, ensure both algorithms have the same starting cash balance.

Different Backtest Parameters

If you don't start your backtest and live deployment on the same date with the same holdings, deviations can occur between backtesting and live trading. To avoid issues, ensure your backtest parameters are the same as your live deployment.

Non-deterministic State From Algorithm Restarts

If you stop and redeploy your live trading algorithm, it needs to restart in a stateful way or else deviations can occur between backtesting and live trading. To avoid issues, redeploy your algorithm in a stateful way using the `SetWarmUp` and `History` methods. Furthermore, use the `ObjectStore` to save state information between your live trading deployments.

Existing Portfolio Securities

If you deploy your algorithm to live trading with a brokerage account that has existing holdings, your live trading equity curve reflects your existing positions, but the backtesting curve won't. Therefore, if you have existing positions in your brokerage account when you deploy your algorithm to live trading, deviations will occur between backtesting and live trading. To avoid issues, deploy your algorithm to live trading using a separate brokerage account or subaccount that does not have existing positions.

Brokerage Limitations

We provide brokerage models that support specific order types and model your buying power. In backtesting, we simulate your orders with the brokerage model you select. In live trading, we send your orders to your brokerage for execution. If the brokerage model that you use in backtesting is not the same brokerage that you use in live trading, deviations may occur between backtesting and live trading. The deviations can occur if your live brokerage doesn't support the order types that you use or if the backtesting brokerage models your buying power with a different methodology than the real brokerage. To avoid brokerage model issues, set the brokerage model in your backtest to the same brokerage that you use in live trading.

Differences From Third-Party Indicators

The values of our indicators can sometimes produce different results than the indicators on other platforms. These discrepancies can be a result of differences in the input data, timestamps, or implementation.

Price Differences

If you find differences in indicator values, compare the prices that are fed into the indicator on each platform. If the input data is slightly different, the indicator values will be different. To test if it's a difference in price data, feed in the data from the third-party platform into our indicators. We validate the indicators against third-party sources and when the values are the same, the indicator values are similar too.

Timestamp Differences

We timestamp our data to the time when the period ends. Many other platforms timestamp to the beginning of the candle.

Implementation Differences

Some indicators can have slightly different default arguments for their implementation. A common difference across platforms is to use a different `MovingAverageType` for the indicators. To view the default arguments for all our

indicators, see Supported Indicators . To view the full implementation of our indicators, see the LEAN GitHub repository .

Warm Up Period Differences

Some platforms use all of the historical data to warm up their indicators while LEAN indicators have a fixed number of bars they need to warm up. As a result, indicators with long memory like the Exponential Moving Average can have slightly different values across platforms.

Differences From Real-time Scheduled Events

In live trading, Scheduled Events execute in a parallel thread based on a real-time clock. If you set a Scheduled Event to fire at 10:00 AM, it executes at exactly 10:00 AM. In backtesting, Scheduled Events are part of the main algorithm manager loop, so they may not execute exactly when you set them. For example, if your algorithm subscribes to minute resolution US Equity data with regular trading hours and you set a Scheduled Event to occur at 2:00 AM, your Scheduled Event will execute at 9:31 AM when the next bar is fed into your algorithm.

The difference between live trading and backtesting is important to note because it can affect your algorithm's behavior. There are two common scenarios to consider.

Execution Timing and Backtest Timeouts

Take the following scenario:

- You set Scheduled Events for 2:00 AM, 3:00 AM, and 4:00 AM each day.
- Each Scheduled Event takes eight minutes to execute.
- Your algorithm only subscribes to US Equity securities without extended market hours (9:30 AM - 4:00 PM).

In this scenario, the Scheduled Events each fire at the correct time and execute without error in live trading. In backtesting, all of the Scheduled Events execute at 9:31 AM when your algorithm receives the first bar of the trading day. Since all the Scheduled Events take eight minutes to execute, the algorithm tries to execute all the Scheduled Events but reaches the 10-minute timeout and the backtest stops execution.

Live Data Delays

In backtests, your algorithm receives data at perfect timing. If you request minute resolution data, your algorithm receives the bars at the top of each minute. In live trading, bars have a slight delay, so you may receive them milliseconds after the top of each minute. Take the following scenario:

- You subscribe to minute resolution data
- You set a Scheduled Event for 10:00 AM
- The Scheduled Event checks the current asset price

In live trading, the Scheduled Event executes at exactly 10:00 AM but your algorithm may receive the 9:59-10:00 AM bar at 10:00:00.01 AM. Therefore, when you check the price in the Scheduled Event, the price from the 9:58-9:59 AM bar is the latest price. In backtesting, the Scheduled Event gets the price from the 9:59-10:00 AM bar since your algorithm receives the bar at perfect timing.

Live Trading

Notifications

Introduction

Set up some live trading notifications so that you are notified of market events and your algorithm's performance. QuantConnect Cloud supports email, SMS, webhooks, and Telegram notifications for live trading algorithms. If you run local algorithm with LEAN or the LEAN CLI, the notification services aren't available. To view the number of notification you can send for free in QuantConnect Cloud, see the Live Trading Notification Quotas .

Email

Email notifications can include up to 10KB of text content in the message body. These notifications can be slow since they go through your email provider. If you don't receive an email notification that you're expecting, check your junk folders.

To send email notifications in your code files, run:

```
self.Notify.Email(email_address, subject, body, attachment, headers)
```

SMS

SMS notifications are the only type of notification that you don't need an internet connection to receive.

To send SMS notifications in your code files, run:

```
self.Notify.Sms(phone_number, message)
```

Webhooks

Webhook notifications are an HTTP-POST request to a URL you provide. The request is sent with a timeout of 300s. You can process these notifications on your web server however you want. For instance, you can inject the content of the notifications into your server's database or use it to create other notifications on your own server.

To send webhook notifications in your code files, run:

```
self.Notify.Web(url, data, headers)
```

Telegram

Telegram notifications are automated messages to a Telegram group.

To send Telegram notifications in your code files, run:

```
self.Notify.Telegram(id, message, token)
```

The `id` argument is your Telegram group Id. Your group Id is in the URL when you open your group chat in the Telegram web interface. For example, the group Id of **web.telegram.org/z/#-503016366** is -503016366.

The `token` optional argument is a token of a bot you must add to your Telegram group. To create a bot, chat with @BotFather and follow its instructions. If you want to use our bot, the username is @quantconnect_notifications_bot.

To add emoticons to your `message` , convert the emoticon character to UTF-32 with the Unicode Converter on the Branah website. For example, the rocket emoji is `0001f680` in UTF-32 format, so use `\U001f680` in your message.

Terms of Use

The notification system can't be used for data distribution.

Live Trading

Charting and Logging

Introduction

The live results page shows your algorithm's live trading performance. You can add custom charts and logs to the results page.

Charting

Custom charts are limited to 4,000 data points. Intensive charting requires hundreds of megabytes of data, which is too much to stream online or display in a web browser. If you exceed the quota, the following message displays:

Exceeded maximum points per chart, data skipped

You can create up to 10 custom chart series per algorithm. If you exceed the quota, your algorithm stops executing and the following message displays:

Exceeded maximum series count: Each backtest can have up to 10 series in total.

In live trading, charts are sampled every one and ten minutes. If you create 1-minute resolution custom charts, the IDE charting will downgrade the granularity and display the 10-minutes sampling after a certain amount of samples.

Logging

Algorithms can record string messages ('log statements') to a file for analysis after a backtest is complete, or as a live algorithm is running. These records can assist in debugging logical flow errors in the project code. Consider adding them in the code block of an `if` statement to signify an error has been caught.

It's good practice to add logging statements to live algorithms so you can understand its behavior and keep records to compare against backtest results. If you don't add logging statements to a live algorithm and the algorithm doesn't trade as you expect, it's difficult to evaluate the underlying problem.

If you run algorithms on QuantConnect, you must stay within the log quota . To only log when your algorithm is live, use the `LiveMode` property.

```py
if self.LiveMode:
    self.Log("My log message")
```

Runtime Statistics

Runtime statistics show the performace of your algorithm at a single moment in time.

The following table describes the default runtime statistics:

Statistic	Description
Equity	The total portfolio value if all of the holdings were sold at current market rates.
Fees	The total quantity of fees paid for all the transactions.
Holdings	The absolute sum of the items in the portfolio.
Net Profit	The dollar-value return across the entire trading period.
PSR	The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).
Return	The rate of return across the entire trading period.
Unrealized	The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.
Volume	The total value of assets traded for all of an algorithm's transactions.

To add a custom runtime statistic, call the `SetRuntimeStatistic` method with a `name` and `value` . The `value` argument can be a `string` or a number.

```
self.SetRuntimeStatistic(name, value)
```

Live Trading

Signal Exports

Our Signal Export integrations let you send live trading signals to third-party services.

CrunchDAO

Equities

Numerai

Equities

See Also

Datasets

Signal Exports

CrunchDAO

Introduction

CrunchDAO is a service that connects strategy developers and capital allocators. With our CrunchDAO integration, you can send trading signals from your live algorithms to the CrunchDAO .

Add Providers

To export signals to CrunchDAO from your algorithm, during initialization , add a CrunchDAO signal export provider.

```py
self.SignalExport.AddSignalExportProviders(CrunchDAOSignalExport(apiKey, model, submissionName, comment))
```

The `CrunchDAOSignalExport` constructor accepts the following arguments:

Argument: `apiKey`

Your CrunchDAO API key. To get your API key, see the Profile > Settings page on the CrunchDAO website.

Data Type: `str` | Default Value: None

Argument: `model`

The Id of your CrunchDAO model. To create a new model or to get the Id of an existing model, see the Profile > Alpha page on the CrunchDAO website.

Data Type: `str` | Default Value: None

Argument: `submissionName`

A name for the submission to distinguish it from your other submissions.

Data Type: `str` | Default Value: `""`

Argument: `comment`

A comment for the submission.

Data Type: `str` | Default Value: `""`

You can add multiple signal export providers to a single algorithm.

Asset Classes

Our CrunchDAO integration supports signals for US Equities .

Universe Selection

The CrunchDAO skeleton contains the universe constituents. To load it into your algorithm, follow these steps:

1. Define a custom universe type .

```
class CrunchDaoSkeleton(PythonData):

    def GetSource(self, config, date, isLive):
        return SubscriptionDataSource("https://tournament.crunchdao.com/data/skeleton.csv",
SubscriptionTransportMedium.RemoteFile)

    def Reader(self, config, line, date, isLive):
        if not line[0].isdigit(): return None
        skeleton = CrunchDaoSkeleton()
        skeleton.Symbol = config.Symbol

        try:
            csv = line.split(',')
            skeleton.EndTime = (datetime.strptime(csv[0], "%Y-%m-%d")).date()
            skeleton.Symbol =  Symbol(SecurityIdentifier.GenerateEquity(csv[1], Market.USA,
mappingResolveDate=skeleton.Time), csv[1])
            skeleton["Ticker"] = csv[1]

        except ValueError:
            # Do nothing
            return None

        return skeleton
```

2. In the `Initialize` method, initialize the universe .

```
self.AddUniverse(CrunchDaoSkeleton, "CrunchDaoSkeleton", Resolution.Daily, self.select_symbols)
```

3. In the algorithm class, define the selector function to select all of the securities in the skeleton file.

```
def select_symbols(self, data: List[CrunchDaoSkeleton]) -> List[Symbol]:
    return [x.Symbol for x in data]
```

Schedule Submissions

CrunchDAO provides a new skeleton every Saturday at 6 PM Coordinated Universal Time (UTC). To schedule your submissions for before the market opens each week, during initialization, you can create a Scheduled Event .

```
self.week = -1
self.Schedule.On(
    self.DateRules.Every([DayOfWeek.Monday, DayOfWeek.Tuesday, DayOfWeek.Wednesday, DayOfWeek.Thursday,
DayOfWeek.Friday]),
    self.TimeRules.At(13, 15, TimeZones.Utc),
    self.submit_signals)
```

The preceding Scheduled Event calls the `submit_signals` method every weekday at 1:15 PM UTC. Define this method to create and send portfolio targets to CrunchDAO.

```py
def submit_signals(self):
    if self.IsWarmingUp:
        return

    # Submit signals once per week
    week_num = self.Time.isocalendar()[1]
    if self.week == week_num:
        return
    self.week = week_num

    symbols = [security.Symbol for security in self.crunch_universe if security.Price > 0]

    # Get historical price data
    # close_prices = self.History(symbols, 22, Resolution.Daily).close.unstack(0)

    # Create portfolio targets
    weight_by_symbol = {symbol: 1/len(symbols) for symbol in symbols} # Add your logic here
    targets = [PortfolioTarget(symbol, weight) for symbol, weight in weight_by_symbol.items()]

    # (Optional) Place trades
    self.SetHoldings(targets)

    # Send signals to CrunchDAO
    success = self.SignalExport.SetTargetPortfolio(targets)
    if not success:
        self.Debug(f"Couldn't send targets at {self.Time}")
```

For more information about requesting historical price and alternative data, see History Requests .

Send Portfolio Targets

To send your current portfolio holdings, call the `SetTargetPortfolioFromPortfolio` method. The method returns a boolean that represents if the targets were successfully sent to CrunchDAO.

```py
success = self.SignalExport.SetTargetPortfolioFromPortfolio()
```

To send targets that aren't based on your current portfolio holdings, pass a `PortfolioTarget` object or a list of `PortfolioTarget` objects to the `SetTargetPortfolio` method. In this situation, the number you pass to the `PortfolioTarget` constructor represents the portfolio weight. Don't use the `PortfolioTarget.Percent` method.

```py
target = PortfolioTarget(self.symbol, weight)
success = self.SignalExport.SetTargetPortfolio(target)
```

Signal Requirements

When you submit signals to CrunchDAO, abide by the following rules:

- All signals must have a quantity between 0 and 1 (inclusive).
- Submit one signal per round. If you submit more than one signal in a round, your new submission overwrites the previous one.
- Send signals for at least half of the investment universe.

Examples

Demonstration Algorithms
CrunchDAOSignalExportDemonstrationAlgorithm.py Python

Signal Exports

Numerai

Introduction

Numerai is a platform where you can earn rewards by uploading unique live trading signals for a universe of US Equities. The regular Numerai Tournament is built around a free dataset they provide , but with Numerai Signals , you can now use any dataset to generate your trading signals. Through our Numerai integration, you can use our rich library of alternative datasets or your own custom datasets to create and automate your Numerai Signal submissions.

Add Providers

To export signals to Numerai from your algorithm, during initialization , add a Numerai signal export provider.

```
self.SignalExport.AddSignalExportProviders(NumeraiSignalExport(publicId, secretId, modelId, fileName))
```

The `NumeraiSignalExport` constructor accepts the following arguments:

Argument: `publicId`

Your Numerai API key. To create and view your API keys, see the **Automation** section of your Account page on the Numerai website.

Data Type: `str` | Default Value: None

Argument: `secretId`

Your Numerai API secret. You get your API secret when you create a new API key.

Data Type: `str` | Default Value: None

Argument: `modelId`

The Id of the Numerai model. Follow these steps to get the model Id:

1. Open the Models page on the Numerai website.
2. In the **Models** section, click the **three dots** next to a model.
3. Click **Copy Model ID** .

Data Type: `str` | Default Value: `""`

Argument: `fileName`

The name for the signal file. If you use a file name that already exists, the new file overwrites the old one.

Data Type: `str` | Default Value: `"predictions.csv"`

You can add multiple signal export providers to a single algorithm.

Asset Classes

Our Numerai integration supports signals for US Equities .

Universe Selection

The Numerai Signals stock market universe covers roughly the top 5,000 largest stocks in the world. The universe available on QuantConnect that's the closest match to the Numerai Signals universe is the CRSP US Total Market Index, which represents approximately 100% of the investable US Equity market regularly traded on the New York Stock Exchange and Nasdaq. This Index doesn't contain all of the stocks in the Numerai Signals universe, but you don't need to submit signals for all the stocks in the Numerai Signals universe.

To get the constituents of the CRSP US Total Market Index, add an ETF constituents universe for the Vanguard Total Stock Market ETF, VTI.

```
self.etf_symbol = self.AddEquity("VTI").Symbol
self.AddUniverse(self.Universe.ETF(self.etf_symbol))
```

Schedule Submissions

Every Tuesday, Wednesday, Thursday, Friday, and Saturday of the week, a new round is open for you to submit signals. Saturday rounds open at 18:00 UTC, and the submission window is open until Monday 14:30 UTC. Weekday rounds open at 13:00 UTC, and the submission window is open for 1 hour. For more information about the competition rounds, see Rounds in the Numerai documentation.

To schedule your submissions, you can set a Scheduled Event during initialization.

```
self.Schedule.On(
    self.DateRules.EveryDay(self.etf_symbol),
    self.TimeRules.At(13, 0, TimeZones.Utc),
    self.submit_signals)
```

Send Portfolio Targets

To send targets, pass a list of `PortfolioTarget` objects to the `SetTargetPortfolio` method. The method returns a boolean that represents if the targets were successfully sent to Numerai Signals. In this situation, the number you pass to the `PortfolioTarget` constructor represents the portfolio weight. Don't use the `PortfolioTarget.Percent` method.

```
success = self.SignalExport.SetTargetPortfolio(targets)
```

Signal Requirements

When you submit signals to Numerai, abide by the following rules:

- All signals must have a quantity between 0 and 1 (exclusive)
- You must submit at least 10 different signals per submission
- At least one of the weights in every batch of signals must be unique

Examples

Demonstration Algorithms
NumeraiSignalExportDemonstrationAlgorithm.py Python

Strategy Library

Introduction

The Strategy Library is a collection of tutorials written by the QuantConnect team and community members. Review these tutorials to learn about trading strategies found in the academic literature and how to implement them with QuantConnect/LEAN.

Tutorials

Search Strategy Tutorials

Strategy Name
CAPM Alpha Ranking Strategy on Dow 30 Companies Applies CAPM model to rank Dow Jones 30 companies.
Combining Mean Reversion and Momentum in Forex Market Combines momentum and mean reversion techniques in the forex markets.
Pairs Trading-Copula vs Cointegration Applies Copula and Cointergration method to pairs trading.
The Dynamic Breakout II Strategy A demonstration of dynamic breakout II strategy.
Dual Thrust Trading Algorithm A demontration of Dual Thrust Intraday strategy.
Can Crude Oil Predict Equity Returns Applies regression method to predict the return from the stock market and compare it to the short-term U.S. T-bill rate.
Intraday Dynamic Pairs Trading using Correlation and Cointegration Approach A high frequency pairs trading algorithm based on cointegration.

Strategy Name
The Momentum Strategy Based on the Low Frequency Component of Forex Market Applies high frequency filter to the momentum strategy.
Stock Selection Strategy Based on Fundamental Factors MorningStar Fundamental factors universe selection algorithm.
Short-Term Reversal Strategy in Stocks A short term reversal algorithm which gives the opposite signal by analyzing recent period price action.
Fundamental Factor Long Short Strategy A basic monthly rebalance long short algorithm based on fundamental factors.
Asset Class Trend Following Selects ETFs over ten-month moving average and assigns an equally weighted allocation. Source: Quantpedia
Asset Class Momentum Selects ETFs in different asset classes with the highest momentum and assigns an equally weighted allocation. Source: Quantpedia
Residual Momentum Constructs a long/short portfolio based on trailing residual momentum normalized by its standard deviation Source: Quantpedia
Sector Momentum Selects ETFs in different sectors with the highest momentum and assigns an equally weighted allocation. Source: Quantpedia
Overnight Anomaly Buy SPY ETF at its closing price and sell it at the opening each day. Source: Quantpedia

Strategy Name
Forex Carry Trade Goes long the currency with the highest central bank interest rate and goes short the currency with the lowest interest rate. Source: Quantpedia
Volatility Effect in Stocks Constructs equally weighted portfolios by selecting stocks with the lowest volatility in the past one year. Source: Quantpedia
Forex Momentum Goes long currencies with strongest 12 month momentum against USD and goes short currencies with the lowest 12 month momentum against USD. Source: Quantpedia
Pairs Trading with Stocks Looks for the security that minimizes the sum of squared deviations and long-short position is opened when pair prices have diverged by multiple of standard deviations. Source: Quantpedia
Short Term Reversal Goes long stocks with the lowest return in the previous month and goes short stocks with the greatest return from the previous month. Source: Quantpedia
Momentum Effect in Stocks Goes long stocks with the best 12-month momentum in the large-cap universe. Source: Quantpedia
Momentum Effect in Country Equity Indexes Goes long stocks with the best 12-month momentum in the country equity indexes ETFs. Source: Quantpedia

Strategy Name
Mean Reversion Effect in Country Equity Indexes Goes long country equity indexes ETFs with the worst 36-month return and short ETFs with the best 36-month return. Source: Quantpedia
Liquidity Effect in Stocks Goes long stocks with the lowest turnover and short on stocks with the highest turnover from the lowest market-cap quartile. Source: Quantpedia
Volatility Risk Premium Effect Sells at-the-money straddle with one month until maturity and buys an offsetting 15% out-of-the-money puts each month. Source: Quantpedia
Momentum Effect in Commodities Futures Goes long commodity futures with the highest momentum and short on futures with the lowest momentum. Source: Quantpedia
Small Capitalization Stocks Premium Anomaly Goes long stocks with the lowest market capitalization and rebalances the portfolio once a year. Source: Quantpedia
Paired Switching Goes long asset with better performance over the last period and rebalances portfolio every quarter. Source: Quantpedia
Term Structure Effect in Commodities Buys each month the 20% of commodities with the highest roll-returns and shorts the 20% of commodities with the lowest roll-returns and holds the long-short positions for one month. Source: Quantpedia

Strategy Name
Momentum Effect Combined with Term Structure in Commodities Portfolios are formed based on roll returns and the algorithm goes long and short contracts with the highest and lowest one-month performance. Source: Quantpedia
Book-to-Market Value Anomaly Quintile portfolios are formed based on the Book-to-Market ratio and the highest quintile is held for one year. Source: Quantpedia
Gold Market Timing Goes long gold when the Fed model shows that the market is undervalued (the earnings yield is higher than the bond yield and their ratio is at least 2). Source: Quantpedia
Turn of the Month in Equity Indexes Buys SPY the day before the end of the month and liquidates position on 3rd trading day of new month. Source: Quantpedia
Momentum - Short Term Reversal Strategy Goes long stocks with the decreasing return from the winner group and short stocks with the increasing return from the loser group. Source: Quantpedia
Pairs Trading with Country ETFs Identifies the price divergence from two highly correlated country ETFs and takes a market neutral position. Source: Quantpedia
Sentiment and Style Rotation Effect in Stocks Creates long-short positions of growth and value stocks based on the investment sentiment. Source: Quantpedia

Strategy Name
Asset Growth Effect Creates long-short positions of stocks based on the annual change of their total assets. Source: Quantpedia
Momentum and State of Market Filters Goes long and short stocks with the highest and lowest six-month momentum respectively if the previous 12 months return on the broad market index was positive. Source: Quantpedia
Accrual Anomaly Decile portfolios are formed based on balance sheet based accruals and highest decile is shorted while lowest decile is bought for a year. Source: Quantpedia
Momentum in Mutual Fund Returns Forms a long-short portfolio of asset management firms based on trailing rate of change and nearness to trailing high. Source: Quantpedia
Momentum and Style Rotation Effect Goes long style index ETF with the highest 12-month momentum and short ETF with the lowest 12-month momentum. Source: Quantpedia
Trading with WTI BRENT Spread Goes long the spread if the spread is below 20-day moving average and short if the spread is above 20-day moving average. Source: Quantpedia

Strategy Name
Momentum Effect in REITs Tercile portfolios are formed based on momentum and the best performing portfolio is held. Source: Quantpedia
Option Expiration Week Effect Goes long S&P 100 index ETF during option expiration week and stays in cash during other days. Source: Quantpedia
Earnings Quality Factor Goes long stocks with high earnings quality and short stocks with low earnings quality based on composite factor score. Source: Quantpedia
January Effect in Stocks Invests into small cap stocks at the beginning of each January and stays invested in large cap stocks for rest of the year. Source: Quantpedia
Momentum and Reversal Combined with Volatility Effect in Stocks Goes long on stocks from the highest performing quintile from the highest volatility group and short on stocks from the lowest performing quintile from the highest volatility group. Source: Quantpedia
ROA Effect within Stocks Goes long on stocks with highest ROA and short stocks with the lowest ROA from each market capitalization group. Source: Quantpedia
January Barometer Invested in equity market with ETF only if January return is positive otherwise switch investments to T-Bills. Source: Quantpedia

Strategy Name
Lunar Cycle in Equity Market Goes long in emerging market index ETF 7 days before the new moon and switch to a short position on emerging market index ETF 7 days before the full moon. Source: Quantpedia
VIX Predicts Stock Index Returns Goes long on equity index ETF if the VIX is in the highest percentile short if VIX is in the lowest percentile in the last two-year history. Source: Quantpedia
Combining Momentum Effect with Volume Goes long stocks with the highest volume from the top momentum decile and short stocks with the highest volume from the bottom momentum decile. Source: Quantpedia
Short Term Reversal with Futures Goes long (short) on futures from the high-volume, low-open interest group with the lowest (greatest) returns in the previous week. Source: Quantpedia
Pre-holiday Effect Invests in equity market 2 days preceding holiday days and stays in cash during the other trading days. Source: Quantpedia
Beta Factors in Stocks Goes long stocks with the bottom beta and short stocks with the top beta, securities are weighted by the ranked betas. Source: Quantpedia

Strategy Name
Exploiting Term Structure of VIX Futures Buys or sells the nearest VIX futures based on the daily roll and hedge against the open positions with E-mini S&P500 futures. Source: Quantpedia
12 Month Cycle in Cross-Section of Stocks Returns Reviews the returns from last January, going long on the top 10% winners and short the bottom 10%. Source: Quantpedia
Momentum Effect in Stocks in Small Portfolios Goes long in the 10 stocks with the highest performance and goes short in the 10 stocks with the lowest performance in the previous one year. Source: Quantpedia
Value Effect within Countries Invests in the cheapest 33% of country ETFs according to CAPE ratios. Source: Quantpedia
Beta Factor in Country Equity Indexes Goes long on the low-beta portfolio and short on the high-beta portfolio in country indexes ETFs. Source: Quantpedia
Price to Earnings Anomaly Invests in stocks with low P/E ratio.
Fama French Five Factors Stock selecting strategy based on Fama-French Five Factors Model. Source: NYU

Strategy Name
Mean-Reversion Statistical Arbitrage Strategy in Stocks Apply statistical arbitrage to take advantage of pricing inefficiencies in stocks. Source: NYU
Expected Idiosyncratic Skewness Stock selection strategy that calculates expected idiosyncratic skewness using Fama-French three-factor model, sorts stocks based on the calculated skewness, and longs the bottom 5%. Source: NYU
Risk Premia in Forex Markets A strategy based on asymmetric tail risks and excess returns in forex markets. Source: NYU
Seasonality Effect based on Same-Calendar Month Returns A strategy that takes long and short positions based on historical same-calendar month returns Source: NYU
Standardized Unexpected Earnings Stock selection strategy that calculates the unexpected earnings, standardizes the unexpected earnings, goes long on the top 5%, and rebalances the portfolio monthly. Source: NYU
Price and Earnings Momentum A momentum strategy based on quarterly returns and earnings growth Source: NYU
Improved Momentum Strategy on Commodities Futures An advanced momentum strategy that modifies the basic momentum strategies by introducing Baltas and Kosowski weights and rebalances the portfolio monthly. The new weighing scheme incorporates trend strength into the trading signal, uses an efficient volatility estimator, and adds a dynamic leverage mechanism. Source: NYU

Strategy Name
Commodities Futures Trend Following A simple trend following strategy on commodities futures. Source: NYU
Forecasting Stock Prices using a Temporal CNN Model Applying a Temporal Convolutional Neural Network to forecasting future stock prices. Source: Tampere University
Leveraged ETFs with Systematic Risk Management We apply Simple Moving Averages to manage the risk of holding leveraged ETFs in an attempt to beat the S&P500 Source: The Lead-Lag Report
Ichimoku Clouds in the Energy Sector A techincal indicator crossover strategy trading the largest energy companies. Source: SSRN
Intraday ETF Momentum A momentum strategy based on returns of the market open Source: NYU
Intraday Arbitrage Between Index ETFs A strategy that tracks the price paths of two correlated ETFs and takes advantage of mis-pricings that arise when the price paths diverge Source: SSRN
Optimal Pairs Trading Mathematically Deriving the Optimal Entry and Liquidation Values of a Pairs Trading Process Source: arXiv

Strategy Name
G-Score Investing Applying G-Score Investing to Invest in a Portfolio of Technology Stocks Source: SSRN
SVM Wavelet Forecasting Forecasting EURJPY prices with an SVM Wavelet model Source: Academia
Gradient Boosting Model Forecasts future intraday returns with a gradient boosting model trained on technical indicators Source: arXiv
Using News Sentiment to Predict Price Direction of Drug Manufacturers Analyzes the news releases of drug manufacturers and places intraday trades for the stocks with positive news. Source: arXiv
Gaussian Naive Bayes Model Forecasts the next day's return of technology stocks by fitting a gaussian naive bayes model to the historical returns of the technology sector constituents. Source: Academia

API Reference

Available QCAlgorithm Methods

All

Adding Data | Algorithm Framework | Charting | Consolidating Data | Handling Data | Historical Data | Indicators | Live Trading | Logging | Machine Learning | Modeling | Parameter and Optimization | Scheduled Events | Securities and Portfolio | Trading and Orders | Universes

ABANDS()	Creates a new Acceleration Bands indicator.
AD()	Creates a new AccumulationDistribution indicator.
AddAlpha()	Adds a new alpha model.
AddCfd()	Creates and adds a new `Cfd` security to the algorithm.
AddChart()	Add a Chart object to algorithm collection.
AddCrypto()	Creates and adds a new `Crypto` security to the algorithm.
AddCryptoFuture()	Creates and adds a new `CryptoFuture` security to the algorithm.
AddData()	AddData a new user defined data source, requiring only the minimum config options. The data is added with a default time zone of NewYork (Eastern Daylight Savings Time). This method is meant for custom data types that require a ticker, but have no underlying

INDICATORS

ABANDS() 1/1

```
AccelerationBands QuantConnect.Algorithm.QCAlgorithm.ABANDS (
Symbol symbol,
Int32 period,
*Decimal width,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Acceleration Bands indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Acceleration Bands we want.
`Int32`	period	The period of the three moving average (middle, upper and lower band).
`*Decimal`	width	*(Optional)* A coefficient specifying the distance between the middle band and upper or lower bands.
`*MovingAverageType`	movingAverageType	*(Optional)* Type of the moving average. *Options: ['Simple', 'Exponential', 'Wilders', 'LinearWeightedMovingAverage', 'DoubleExponential', 'TripleExponential', 'Triangular', 'T3', 'Kama', 'Hull', 'Alma']*
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AccelerationBands` - The new `AccelerationBands` object.

Definition at line 46 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

AD() 1/1

```
AccumulationDistribution QuantConnect.Algorithm.QCAlgorithm.AD (
 Symbol symbol,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new AccumulationDistribution indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose AD we want.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AccumulationDistribution` - The AccumulationDistribution indicator for the requested symbol over the specified period.

Definition at line 64 of file Algorithm/QCAlgorithm.Indicators.cs.

ALGORITHM FRAMEWORK

AddAlpha() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddAlpha (
IAlphaModel alpha
)
```

Adds a new alpha model.

Show Details

Parameters		
`IAlphaModel`	alpha	Model that generates alpha to add.

Return

`Void` - This method provides no return.

Definition at line 326 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK

AddAlpha() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddAlpha (
PyObject alpha
)
```

Adds a new alpha model.

Show Details

Parameters		
PyObject	alpha	Model that generates alpha to add.

Return

`Void` - This method provides no return.

Definition at line 49 of file Algorithm/QCAlgorithm.Framework.Python.cs.

ADDING DATA

AddCfd() 1/1

```
Cfd QuantConnect.Algorithm.QCAlgorithm.AddCfd (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage
)
```

Creates and adds a new `Cfd` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The currency pair.
`*Nullable< Resolution >`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The cfd trading market, See also: `Market` . . Default value is None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .

Return

`Cfd` - The new security.

Definition at line 2167 of file Algorithm/QCAlgorithm.cs.

CHARTING

AddChart() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.AddChart (
Chart chart
)
```

Add a Chart object to algorithm collection.

Show Details

Parameters		
`Chart`	chart	Chart object to add to collection.

Return

`Void` - This method provides no return.

Definition at line 53 of file Algorithm/QCAlgorithm.Plotting.cs.

ADDING DATA

AddCrypto() 1/1

```
Crypto QuantConnect.Algorithm.QCAlgorithm.AddCrypto (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage
)
```

Creates and adds a new `Crypto` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The currency pair.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The cfd trading market, See also: `Market` . . Default value is None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .

Return

`Crypto` - The new security.

Definition at line 2198 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddCryptoFuture() 1/1

```
CryptoFuture QuantConnect.Algorithm.QCAlgorithm.AddCryptoFuture (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage
)
```

Creates and adds a new `CryptoFuture` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The currency pair.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The cfd trading market, See also: `Market` . . Default value is None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .

Return

`CryptoFuture` - The new security.

Definition at line 2213 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 1/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
PyObject type,
String ticker,
*Nullable<Resolution> resolution
)
```

AddData a new user defined data source, requiring only the minimum config options. The data is added with a default time zone of NewYork (Eastern Daylight Savings Time). This method is meant for custom data types that require a ticker, but have no underlying Symbol. Examples of data sources that meet this criteria are U.S. Treasury Yield Curve Rates and Trading Economics data.

Show Details

Parameters		
`PyObject`	type	Data source type.
`String`	ticker	Key/Ticker for data.
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the data.

Return

`Security` - The new `Security` object.

Definition at line 62 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 2/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
PyObject type,
Symbol underlying,
*Nullable<Resolution> resolution
)
```

AddData a new user defined data source, requiring only the minimum config options. The data is added with a default time zone of NewYork (Eastern Daylight Savings Time). This adds a Symbol to the `Underlying` property in the custom data Symbol object. Use this method when adding custom data with a ticker from the past, such as "AOL" before it became "TWX", or if you need to filter using custom data and place trades on the Symbol associated with the custom data.

Show Details

Parameters		
`PyObject`	type	Data source type.
`Symbol`	underlying	The underlying symbol for the custom data.
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the data.

Return

`Security` - The new `Security` object.

Definition at line 86 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 3/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
PyObject type,
String ticker,
Nullable<Resolution> resolution,
DateTimeZone timeZone,
*Boolean fillForward,
*Decimal leverage
)
```

AddData a new user defined data source, requiring only the minimum config options. This method is meant for custom data types that require a ticker, but have no underlying Symbol. Examples of data sources that meet this criteria are U.S. Treasury Yield Curve Rates and Trading Economics data.

Show Details

Parameters		
PyObject	type	Data source type.
String	ticker	Key/Ticker for data.
Nullable<Resolution>	resolution	Resolution of the Data Required.
DateTimeZone	timeZone	Specifies the time zone of the raw data.
*Boolean	fillForward	*(Optional)* When no data available on a tradebar, return the last data that was generated.
*Decimal	leverage	*(Optional)* Custom leverage per security.

Return

`Security` - The new `Security` object.

Definition at line 104 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 4/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
PyObject type,
Symbol underlying,
Nullable<Resolution> resolution,
DateTimeZone timeZone,
*Boolean fillForward,
*Decimal leverage
)
```

AddData a new user defined data source, requiring only the minimum config options. This adds a Symbol to the `Underlying` property in the custom data Symbol object. Use this method when adding custom data with a ticker from the past, such as "AOL" before it became "TWX", or if you need to filter using custom data and place trades on the Symbol associated with the custom data.

Show Details

Parameters		
PyObject	type	Data source type.
Symbol	underlying	The underlying symbol for the custom data.
Nullable<Resolution>	resolution	Resolution of the Data Required.
DateTimeZone	timeZone	Specifies the time zone of the raw data.
*Boolean	fillForward	*(Optional)* When no data available on a tradebar, return the last data that was generated.
*Decimal	leverage	*(Optional)* Custom leverage per security.

Return

`Security` - The new `Security` object.

Definition at line 130 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 5/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
Type dataType,
String ticker,
Nullable<Resolution> resolution,
DateTimeZone timeZone,
*Boolean fillForward,
*Decimal leverage
)
```

AddData a new user defined data source, requiring only the minimum config options. This method is meant for custom data types that require a ticker, but have no underlying Symbol. Examples of data sources that meet this criteria are U.S. Treasury Yield Curve Rates and Trading Economics data.

Show Details

Parameters		
`Type`	dataType	Data source type.
`String`	ticker	Key/Ticker for data.
`Nullable<Resolution>`	resolution	Resolution of the Data Required.
`DateTimeZone`	timeZone	Specifies the time zone of the raw data.
`*Boolean`	fillForward	*(Optional)* When no data available on a tradebar, return the last data that was generated.
`*Decimal`	leverage	*(Optional)* Custom leverage per security.

Return

`Security` - The new `Security` object.

Definition at line 148 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 6/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
Type dataType,
Symbol underlying,
*Nullable<Resolution> resolution,
*DateTimeZone timeZone,
*Boolean fillForward,
*Decimal leverage
)
```

AddData a new user defined data source, requiring only the minimum config options. This adds a Symbol to the `Underlying` property in the custom data Symbol object. Use this method when adding custom data with a ticker from the past, such as "AOL" before it became "TWX", or if you need to filter using custom data and place trades on the Symbol associated with the custom data.

Show Details

Parameters		
`Type`	dataType	Data source type.
`Symbol`	underlying	/
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the Data Required.
`*DateTimeZone`	timeZone	*(Optional)* Specifies the time zone of the raw data.
`*Boolean`	fillForward	*(Optional)* When no data available on a tradebar, return the last data that was generated.
`*Decimal`	leverage	*(Optional)* Custom leverage per security.

Return

`Security` - The new `Security` object.

Definition at line 193 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 7/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
PyObject type,
String ticker,
SymbolProperties properties,
SecurityExchangeHours exchangeHours,
*Nullable<Resolution> resolution,
*Boolean fillForward,
*Decimal leverage
)
```

AddData a new user defined data source including symbol properties and exchange hours, all other vars are not required and will use defaults. This overload reflects the C# equivalent for custom properties and market hours.

Show Details

Parameters		
PyObject	type	Data source type.
String	ticker	Key/Ticker for data.
SymbolProperties	properties	The properties of this new custom data.
SecurityExchangeHours	exchangeHours	The Exchange hours of this symbol.
*Nullable<Resolution>	resolution	*(Optional)* Resolution of the Data Required.
*Boolean	fillForward	*(Optional)* When no data available on a tradebar, return the last data that was generated.
*Decimal	leverage	*(Optional)* Custom leverage per security.

Return

Security - The new Security object.

Definition at line 213 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddData() 8/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
 String ticker,
 *Nullable<Resolution> resolution
)
```

Show Details

Parameters		
String	ticker	/
*Nullable<Resolution>	resolution	*(Optional)* /

Return

Security - The new Security object.

Definition at line 2323 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 9/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
 Symbol underlying,
 *Nullable<Resolution> resolution
)
```

Show Details

Parameters		
Symbol	underlying	/
*Nullable<Resolution>	resolution	*(Optional)* /

Return

Security - The new Security object.

Definition at line 2342 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 10/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
 String ticker,
 Nullable<Resolution> resolution,
 Boolean fillForward,
 *Decimal leverage
)
```

Show Details

Parameters		
String	ticker	/
Nullable<Resolution>	resolution	/
Boolean	fillForward	/
*Decimal	leverage	*(Optional)* /

Return

Security - The new Security object.

Definition at line 2364 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 11/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
 Symbol underlying,
 Nullable<Resolution> resolution,
 Boolean fillForward,
 *Decimal leverage
)
```

Show Details

Parameters		
Symbol	underlying	/
Nullable<Resolution>	resolution	/
Boolean	fillForward	/
*Decimal	leverage	*(Optional)* /

Return

Security - The new Security object.

Definition at line 2381 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 12/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
String ticker,
Nullable<Resolution> resolution,
DateTimeZone timeZone,
*Boolean fillForward,
*Decimal leverage
)
```

Show Details

Parameters		
String	ticker	/
Nullable<Resolution>	resolution	/
DateTimeZone	timeZone	/
*Boolean	fillForward	*(Optional)* /
*Decimal	leverage	*(Optional)* /

Return

Security - The new Security object.

Definition at line 2398 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 13/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
Symbol underlying,
Nullable<Resolution> resolution,
DateTimeZone timeZone,
*Boolean fillForward,
*Decimal leverage
)
```

Show Details

Parameters		
`Symbol`	underlying	/
`Nullable<Resolution>`	resolution	/
`DateTimeZone`	timeZone	/
`*Boolean`	fillForward	*(Optional)* /
`*Decimal`	leverage	*(Optional)* /

Return

`Security` - The new `Security` object.

Definition at line 2415 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddData() 14/14

```
Security QuantConnect.Algorithm.QCAlgorithm.AddData (
String ticker,
SymbolProperties properties,
SecurityExchangeHours exchangeHours,
*Nullable<Resolution> resolution,
*Boolean fillForward,
*Decimal leverage
)
```

Show Details

Parameters		
`String`	ticker	/
`SymbolProperties`	properties	/
`SecurityExchangeHours`	exchangeHours	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Boolean`	fillForward	*(Optional)* /
`*Decimal`	leverage	*(Optional)* /

Return

`Security` - The new `Security` object.

Definition at line 2433 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddEquity() 1/1

```
Equity QuantConnect.Algorithm.QCAlgorithm.AddEquity (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage,
*Boolean extendedMarketHours,
*Nullable<DataNormalizationMode> dataNormalizationMode
)
```

Creates and adds a new `Equity` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The equity ticker symbol.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The equity's market, See also: `Market` . . Default value is None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .
`*Boolean`	extendedMarketHours	*(Optional)* True to send data during pre and post market sessions. Default is False.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the equity.

Return

`Equity` - The new security.

Definition at line 1796 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddForex() 1/1

```
Forex QuantConnect.Algorithm.QCAlgorithm.AddForex (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage
)
```

Creates and adds a new `Forex` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The currency pair.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The foreign exchange trading market, See also: `Market` . . Default value is None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .

Return

`Forex` - The new security.

Definition at line 2152 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddFuture() 1/1

```
Future QuantConnect.Algorithm.QCAlgorithm.AddFuture (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage,
*Boolean extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Int32 contractDepthOffset
)
```

Creates and adds a new `Future` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The future ticker.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The futures market, See also: `Market` . . Default is value None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .
`*Boolean`	extendedMarketHours	*(Optional)* Use extended market hours data.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the continuous future contract.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the continuous future contract.
`*Int32`	contractDepthOffset	*(Optional)* The continuous future contract desired offset from the current front month. For example, 0 (default) will use the front month, 1 will use the back month contract.

Return

`Future` - The new security.

Definition at line 1906 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddFutureContract() 1/1

```
Future QuantConnect.Algorithm.QCAlgorithm.AddFutureContract (
 Symbol symbol,
 *Nullable<Resolution> resolution,
 *Boolean fillForward,
 *Decimal leverage,
 *Boolean extendedMarketHours
)
```

Creates and adds a new single `Future` contract to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	The futures contract symbol.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .
`*Boolean`	extendedMarketHours	*(Optional)* Use extended market hours data.

Return

`Future` - The new security.

Definition at line 1942 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddFutureOption() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddFutureOption (
Symbol futureSymbol,
PyObject optionFilter
)
```

Creates and adds a new Future Option contract to the algorithm.

Show Details

Parameters		
Symbol	futureSymbol	The Future canonical symbol (i.e. Symbol returned from AddFuture).
PyObject	optionFilter	Filter to apply to option contracts loaded as part of the universe.

Return

Void - This method provides no return.

Definition at line 234 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA

AddFutureOption() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddFutureOption (
Symbol symbol,
*Func<OptionFilterUniverse, OptionFilterUniverse> optionFilter
)
```

Creates and adds a new Future Option contract to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	The `Future` canonical symbol (i.e. Symbol returned from `AddFuture`).
`*Func<OptionFilterUniverse, OptionFilterUniverse>`	optionFilter	*(Optional)* Filter to apply to option contracts loaded as part of the universe.

Return

`Void` - This method provides no return.

Definition at line 1956 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddFutureOptionContract() 1/1

```
Option QuantConnect.Algorithm.QCAlgorithm.AddFutureOptionContract (
Symbol symbol,
*Nullable<Resolution> resolution,
*Boolean fillForward,
*Decimal leverage,
*Boolean extendedMarketHours
)
```

Adds a future option contract to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	Option contract Symbol.
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the option contract, i.e. the granularity of the data.
`*Boolean`	fillForward	*(Optional)* If True, this will fill in missing data points with the previous data point.
`*Decimal`	leverage	*(Optional)* The leverage to apply to the option contract.
`*Boolean`	extendedMarketHours	*(Optional)* Use extended market hours data.

Return

`Option` - Option security.

Definition at line 1977 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddIndex() 1/1

```
Index QuantConnect.Algorithm.QCAlgorithm.AddIndex (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward
)
```

Creates and adds a new `Index` security to the algorithm.

Show Details

Parameters		
`String`	ticker	The currency pair.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The index trading market, See also: `Market` . . Default value is None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.

Return

`Index` - The new security.

Definition at line 2182 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddIndexOption() 1/3

```
Option QuantConnect.Algorithm.QCAlgorithm.AddIndexOption (
String ticker,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward
)
```

Creates and adds index options to the algorithm.

Show Details

Parameters		
String	ticker	The ticker of the Index Option.
*Nullable<Resolution>	resolution	*(Optional)* Resolution of the index option contracts, i.e. the granularity of the data.
*String	market	*(Optional)* Market of the index option. If no market is provided, we default to USA .
*Boolean	fillForward	*(Optional)* If True, this will fill in missing data points with the previous data point.

Return

Option - Canonical Option security.

Definition at line 1997 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddIndexOption() 2/3

```
Option QuantConnect.Algorithm.QCAlgorithm.AddIndexOption (
Symbol symbol,
*Nullable<Resolution> resolution,
*Boolean fillForward
)
```

Creates and adds index options to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	The Symbol of the `Security` returned from .
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the index option contracts, i.e. the granularity of the data.
`*Boolean`	fillForward	*(Optional)* If True, this will fill in missing data points with the previous data point.

Return

`Option` - Canonical Option security.

Definition at line 2013 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddIndexOption() 3/3

```
Option QuantConnect.Algorithm.QCAlgorithm.AddIndexOption (
Symbol symbol,
String targetOption,
*Nullable<Resolution> resolution,
*Boolean fillForward
)
```

Creates and adds index options to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	The Symbol of the `Security` returned from .
`String`	targetOption	The target option ticker. This is useful when the option ticker does not match the underlying, e.g. SPX index and the SPXW weekly option. If None is provided will use underlying.
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the index option contracts, i.e. the granularity of the data.
`*Boolean`	fillForward	*(Optional)* If True, this will fill in missing data points with the previous data point.

Return

`Option` - Canonical Option security.

Definition at line 2027 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddIndexOptionContract() 1/1

```
Option QuantConnect.Algorithm.QCAlgorithm.AddIndexOptionContract (
Symbol symbol,
*Nullable<Resolution> resolution,
*Boolean fillForward
)
```

Adds an index option contract to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	Symbol of the index option contract.
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the index option contract, i.e. the granularity of the data.
`*Boolean`	fillForward	*(Optional)* If True, this will fill in missing data points with the previous data point.

Return

`Option` - Index Option Contract.

Definition at line 2046 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddOption() 1/3

```
Option QuantConnect.Algorithm.QCAlgorithm.AddOption (
String underlying,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage
)
```

Creates and adds a new equity `Option` security to the algorithm.

Show Details

Parameters		
`String`	underlying	The underlying equity ticker.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The equity's market, See also: `Market` . . Default is value None and looked up using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .

Return

`Option` - The new security.

Definition at line 1812 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddOption() 2/3

```
Option QuantConnect.Algorithm.QCAlgorithm.AddOption (
Symbol underlying,
*Nullable<Resolution> resolution,
*String market,
*Boolean fillForward,
*Decimal leverage
)
```

Creates and adds a new `Option` security to the algorithm. This method can be used to add options with non-equity asset classes to the algorithm (e.g. Future Options).

Show Details

Parameters		
`Symbol`	underlying	Underlying asset Symbol to use as the option's underlying.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*String`	market	*(Optional)* The option's market, See also: `Market` . . Default value is None, but will be resolved using BrokerageModel.DefaultMarkets in `AddSecurity<T>` .
`*Boolean`	fillForward	*(Optional)* If True, data will be provided to the algorithm every Second, Minute, Hour, or Day, while the asset is open and depending on the Resolution this option was configured to use.
`*Decimal`	leverage	*(Optional)* The requested leverage for the .

Return

`Option` - The new option security instance.

Definition at line 1839 of file Algorithm/QCAlgorithm.cs.

<-- Missing documentation attribute for AddOption -->

ADDING DATA

AddOptionContract() 1/1

```
Option QuantConnect.Algorithm.QCAlgorithm.AddOptionContract (
Symbol symbol,
*Nullable<Resolution> resolution,
*Boolean fillForward,
*Decimal leverage,
*Boolean extendedMarketHours
)
```

Creates and adds a new single `Option` contract to the algorithm.

Show Details

Parameters		
`Symbol`	symbol	The option contract symbol.
`*Nullable<Resolution>`	resolution	*(Optional)* The `Resolution` of market data, Tick, Second, Minute, Hour, or Daily. Default is `Minute` .
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice. Default is True.
`*Decimal`	leverage	*(Optional)* The requested leverage for this equity. Default is set by `SecurityInitializer` .
`*Boolean`	extendedMarketHours	*(Optional)* Use extended market hours data.

Return

`Option` - The new security.

Definition at line 2066 of file Algorithm/QCAlgorithm.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

AddRiskManagement() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddRiskManagement (
 IRiskManagementModel riskManagement
)
```

Adds a new risk management model.

Show Details

Parameters		
`IRiskManagementModel`	riskManagement	Model defining how risk is managed to add.

Return

`Void` - This method provides no return.

Definition at line 384 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

AddRiskManagement() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddRiskManagement (
PyObject riskManagement
)
```

Adds a new risk management model.

Show Details

Parameters		
PyObject	riskManagement	Model defining how risk is managed to add.

Return

Void - This method provides no return.

Definition at line 156 of file Algorithm/QCAlgorithm.Framework.Python.cs.

ADDING DATA

AddSecurity() 1/4

```
Security QuantConnect.Algorithm.QCAlgorithm.AddSecurity (
SecurityType securityType,
String ticker,
*Nullable<Resolution> resolution,
*Boolean fillForward,
*Boolean extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode
)
```

Add specified data to our data subscriptions. QuantConnect will funnel this data to the handle data routine.

Show Details

Parameters		
SecurityType	securityType	MarketType Type: Equity, Commodity, Future, FOREX or Crypto. *Options: ['Base', 'Equity', 'Option', 'Commodity', 'Forex', 'Future', 'Cfd', 'Crypto', 'FutureOption', 'Index', 'IndexOption', 'CryptoFuture']*
String	ticker	The security ticker.
*Nullable<Resolution>	resolution	*(Optional)* Resolution of the Data Required.
*Boolean	fillForward	*(Optional)* When no data available on a tradebar, return the last data that was generated.
*Boolean	extendedMarketHours	*(Optional)* Use extended market hours data.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the security.

Return

`Security` - The new `Security` object.

Definition at line 1597 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddSecurity() 2/4

```
Security QuantConnect.Algorithm.QCAlgorithm.AddSecurity (
SecurityType securityType,
String ticker,
Nullable<Resolution> resolution,
Boolean fillForward,
Decimal leverage,
Boolean extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode
)
```

Add specified data to required list. QC will funnel this data to the handle data routine.

Show Details

Parameters		
`SecurityType`	securityType	MarketType Type: Equity, Commodity, Future, FOREX or Crypto.
`String`	ticker	The security ticker.
`Nullable<Resolution>`	resolution	Resolution of the Data Required.
`Boolean`	fillForward	When no data available on a tradebar, return the last data that was generated.
`Decimal`	leverage	Custom leverage per security.
`Boolean`	extendedMarketHours	Use extended market hours data.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the security.

Return

`Security` - The new `Security` object.

Definition at line 1616 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddSecurity() 3/4

```
Security QuantConnect.Algorithm.QCAlgorithm.AddSecurity (
SecurityType securityType,
String ticker,
Nullable<Resolution> resolution,
String market,
Boolean fillForward,
Decimal leverage,
Boolean extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode
)
```

Set a required SecurityType-symbol and resolution for algorithm.

Show Details

Parameters		
`SecurityType`	securityType	MarketType Type: Equity, Commodity, Future, FOREX or Crypto.
`String`	ticker	The security ticker, e.g. AAPL.
`Nullable<Resolution>`	resolution	Resolution of the MarketType required: MarketData, Second or Minute.
`String`	market	The market the requested security belongs to, such as 'usa' or 'fxcm'.
`Boolean`	fillForward	If True, returns the last available data even if none in that timeslice.
`Decimal`	leverage	leverage for this security.
`Boolean`	extendedMarketHours	Use extended market hours data.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the security.

Return

`Security` - The new `Security` object.

Definition at line 1635 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

AddSecurity() 4/4

```
Security QuantConnect.Algorithm.QCAlgorithm.AddSecurity (
Symbol symbol,
*Nullable<Resolution> resolution,
*Boolean fillForward,
*Decimal leverage,
*Boolean extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Int32 contractDepthOffset
)
```

Set a required SecurityType-symbol and resolution for algorithm.

Show Details

Parameters		
`Symbol`	symbol	The security Symbol.
`*Nullable<Resolution>`	resolution	*(Optional)* Resolution of the MarketType required: MarketData, Second or Minute.
`*Boolean`	fillForward	*(Optional)* If True, returns the last available data even if none in that timeslice.
`*Decimal`	leverage	*(Optional)* leverage for this security.
`*Boolean`	extendedMarketHours	*(Optional)* Use extended market hours data.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the security.
`*Int32`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 (default) will use the front month, 1 will use the back month contract.

Return

`Security` - The new `Security` object.

Definition at line 1690 of file Algorithm/QCAlgorithm.cs.

CHARTING

AddSeries() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.AddSeries (
String chart,
String series,
SeriesType seriesType,
*String unit
)
```

Add a series object for charting. This is useful when initializing charts with series other than type = line. If a series exists in the chart with the same name, then it is replaced.

Show Details

Parameters		
`String`	chart	The chart name.
`String`	series	The series name.
`SeriesType`	seriesType	The type of series, i.e, Scatter. *Options: ['Line', 'Scatter', 'Candle', 'Bar', 'Flag', 'StackedArea', 'Pie', 'Treemap']*
`*String`	unit	*(Optional)* The unit of the y axis, usually $.

Return

`Void` - This method provides no return.

Definition at line 231 of file Algorithm/QCAlgorithm.Plotting.cs.

UNIVERSES

AddUniverse() 1/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
SecurityType securityType,
String name,
Resolution resolution,
String market,
UniverseSettings universeSettings,
Func<IEnumerable<T>, IEnumerable<Symbol>> selector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
SecurityType	securityType	/
String	name	/
Resolution	resolution	/
String	market	/
UniverseSettings	universeSettings	/
Func<IEnumerable<T>, IEnumerable<Symbol>>	selector	/

Return

Universe - The new Universe object.

Definition at line 379 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 2/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
SecurityType securityType,
String name,
Resolution resolution,
String market,
UniverseSettings universeSettings,
Func<IEnumerable<T>, IEnumerable<String>> selector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
`SecurityType`	securityType	/
`String`	name	/
`Resolution`	resolution	/
`String`	market	/
`UniverseSettings`	universeSettings	/
`Func<IEnumerable<T>, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 400 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 3/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
Func<IEnumerable<CoarseFundamental>, IEnumerable<Symbol>> selector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
`Func<IEnumerable<CoarseFundamental>, IEnumerable<Symbol>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 418 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 4/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
Func<IEnumerable<CoarseFundamental>, IEnumerable<Symbol>> coarseSelector,
Func<IEnumerable<FineFundamental>, IEnumerable<Symbol>> fineSelector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
Func<IEnumerable<CoarseFundamental>, IEnumerable<Symbol>>	coarseSelector	/
Func<IEnumerable<FineFundamental>, IEnumerable<Symbol>>	fineSelector	/

Return

Universe - The new Universe object.

Definition at line 430 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 5/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
Universe universe,
Func<IEnumerable<FineFundamental>, IEnumerable<Symbol>> fineSelector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
Universe	universe	/
Func<IEnumerable<FineFundamental>, IEnumerable<Symbol>>	fineSelector	/

Return

`Universe` - The new `Universe` object.

Definition at line 444 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 6/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Func<DateTime, IEnumerable<String>> selector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
`String`	name	/
`Func<DateTime, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 456 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 7/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Resolution resolution,
Func<DateTime, IEnumerable<String>> selector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
`String`	name	/
`Resolution`	resolution	/
`Func<DateTime, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 469 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 8/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
SecurityType securityType,
String name,
Resolution resolution,
String market,
UniverseSettings universeSettings,
Func<DateTime, IEnumerable<String>> selector
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
`SecurityType`	securityType	/
`String`	name	/
`Resolution`	resolution	/
`String`	market	/
`UniverseSettings`	universeSettings	/
`Func<DateTime, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 484 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 9/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
Universe universe
)
```

Adds the universe to the algorithm.

Show Details

Parameters		
`Universe`	universe	The universe to be added.

Return

`Universe` - The new `Universe` object.

Definition at line 204 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 10/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Func<IEnumerable<T>, IEnumerable<Symbol>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`Func<IEnumerable<T>, IEnumerable<Symbol>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 225 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 11/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
 String name,
 Func<IEnumerable<T>, IEnumerable<String>> selector
 )
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`Func<IEnumerable<T>, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 239 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 12/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
UniverseSettings universeSettings,
Func<IEnumerable<T>, IEnumerable<Symbol>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`UniverseSettings`	universeSettings	/
`Func<IEnumerable<T>, IEnumerable<Symbol>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 254 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 13/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
UniverseSettings universeSettings,
Func<IEnumerable<T>, IEnumerable<String>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`UniverseSettings`	universeSettings	/
`Func<IEnumerable<T>, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 269 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 14/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
 String name,
 Resolution resolution,
 Func<IEnumerable<T>, IEnumerable<Symbol>> selector
 )
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`Resolution`	resolution	/
`Func<IEnumerable<T>, IEnumerable<Symbol>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 284 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 15/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Resolution resolution,
Func<IEnumerable<T>, IEnumerable<String>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`Resolution`	resolution	/
`Func<IEnumerable<T>, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 299 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 16/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Resolution resolution,
UniverseSettings universeSettings,
Func<IEnumerable<T>, IEnumerable<Symbol>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`Resolution`	resolution	/
`UniverseSettings`	universeSettings	/
`Func<IEnumerable<T>, IEnumerable<Symbol>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 315 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 17/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Resolution resolution,
UniverseSettings universeSettings,
Func<IEnumerable<T>, IEnumerable<String>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`String`	name	/
`Resolution`	resolution	/
`UniverseSettings`	universeSettings	/
`Func<IEnumerable<T>, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 331 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 18/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
SecurityType securityType,
String name,
Resolution resolution,
String market,
Func<IEnumerable<T>, IEnumerable<Symbol>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`SecurityType`	securityType	/
`String`	name	/
`Resolution`	resolution	/
`String`	market	/
`Func<IEnumerable<T>, IEnumerable<Symbol>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 347 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 19/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
SecurityType securityType,
String name,
Resolution resolution,
String market,
Func<IEnumerable<T>, IEnumerable<String>> selector
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
`SecurityType`	securityType	/
`String`	name	/
`Resolution`	resolution	/
`String`	market	/
`Func<IEnumerable<T>, IEnumerable<String>>`	selector	/

Return

`Universe` - The new `Universe` object.

Definition at line 363 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverse() 20/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
PyObject pyObject
)
```

Creates a new universe and adds it to the algorithm. This is for coarse fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
PyObject	pyObject	Defines an initial coarse selection.

Return

`Universe` - The new `Universe` object.

Definition at line 287 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 21/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
 PyObject pyObject,
 PyObject pyfine
)
```

Creates a new universe and adds it to the algorithm. This is for coarse and fine fundamental US Equity data and will be executed on day changes in the NewYork time zone (`NewYork` .

Show Details

Parameters		
PyObject	pyObject	Defines an initial coarse selection or a universe.
PyObject	pyfine	Defines a more detailed selection with access to more data.

Return

`Universe` - The new `Universe` object.

Definition at line 321 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 22/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
Resolution resolution,
PyObject pySelector
)
```

Creates a new universe and adds it to the algorithm. This can be used to return a list of string symbols retrieved from anywhere and will loads those symbols under the US Equity market.

Show Details

Parameters		
String	name	A unique name for this universe.
Resolution	resolution	The resolution this universe should be triggered on.
PyObject	pySelector	Function delegate that accepts a DateTime and returns a collection of string symbols.

Return

Universe - The new Universe object.

Definition at line 353 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 23/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
String name,
PyObject pySelector
)
```

Creates a new universe and adds it to the algorithm. This can be used to return a list of string symbols retrieved from anywhere and will loads those symbols under the US Equity market.

Show Details

Parameters		
String	name	A unique name for this universe.
PyObject	pySelector	Function delegate that accepts a DateTime and returns a collection of string symbols.

Return

`Universe` - The new `Universe` object.

Definition at line 366 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 24/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
 SecurityType securityType,
 String name,
 Resolution resolution,
 String market,
 UniverseSettings universeSettings,
 PyObject pySelector
)
```

Creates a new user defined universe that will fire on the requested resolution during market hours.

Show Details

Parameters		
SecurityType	securityType	The security type of the universe.
String	name	A unique name for this universe.
Resolution	resolution	The resolution this universe should be triggered on.
String	market	The market of the universe.
UniverseSettings	universeSettings	The subscription settings used for securities added from this universe.
PyObject	pySelector	Function delegate that accepts a DateTime and returns a collection of string symbols.

Return

`Universe` - The new `Universe` object.

Definition at line 382 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 25/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
PyObject T,
String name,
PyObject selector
)
```

Creates a new universe and adds it to the algorithm. This will use the default universe settings specified via the `UniverseSettings` property. This universe will use the defaults of SecurityType.Equity, Resolution.Daily, Market.USA, and UniverseSettings.

Show Details

Parameters		
`PyObject`	T	The data type.
`String`	name	A unique name for this universe.
`PyObject`	selector	Function delegate that performs selection on the universe data.

Return

`Universe` - The new `Universe` object.

Definition at line 397 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 26/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
PyObject T,
String name,
Resolution resolution,
PyObject selector
)
```

Creates a new universe and adds it to the algorithm. This will use the default universe settings specified via the `UniverseSettings` property. This universe will use the defaults of SecurityType.Equity, Market.USA and UniverseSettings.

Show Details

Parameters		
`PyObject`	T	The data type.
`String`	name	A unique name for this universe.
`Resolution`	resolution	The expected resolution of the universe data.
`PyObject`	selector	Function delegate that performs selection on the universe data.

Return

`Universe` - The new `Universe` object.

Definition at line 412 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 27/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
PyObject T,
String name,
Resolution resolution,
UniverseSettings universeSettings,
PyObject selector
)
```

Creates a new universe and adds it to the algorithm. This will use the default universe settings specified via the `UniverseSettings` property. This universe will use the defaults of SecurityType.Equity, and Market.USA.

Show Details

Parameters		
PyObject	T	The data type.
String	name	A unique name for this universe.
Resolution	resolution	The expected resolution of the universe data.
UniverseSettings	universeSettings	The settings used for securities added by this universe.
PyObject	selector	Function delegate that performs selection on the universe data.

Return

Universe - The new Universe object.

Definition at line 428 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 28/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
 PyObject T,
 String name,
 UniverseSettings universeSettings,
 PyObject selector
)
```

Creates a new universe and adds it to the algorithm. This will use the default universe settings specified via the UniverseSettings property. This universe will use the defaults of SecurityType.Equity, Resolution.Daily, and Market.USA.

Show Details

Parameters		
PyObject	T	The data type.
String	name	A unique name for this universe.
UniverseSettings	universeSettings	The settings used for securities added by this universe.
PyObject	selector	Function delegate that performs selection on the universe data.

Return

`Universe` - The new `Universe` object.

Definition at line 443 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 29/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
PyObject T,
SecurityType securityType,
String name,
Resolution resolution,
String market,
PyObject selector
)
```

Creates a new universe and adds it to the algorithm. This will use the default universe settings specified via the `UniverseSettings` property.

Show Details

Parameters		
`PyObject`	T	The data type.
`SecurityType`	securityType	The security type the universe produces.
`String`	name	A unique name for this universe.
`Resolution`	resolution	The expected resolution of the universe data.
`String`	market	The market for selected symbols.
`PyObject`	selector	Function delegate that performs selection on the universe data.

Return

`Universe` - The new `Universe` object.

Definition at line 459 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 30/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
PyObject T,
SecurityType securityType,
String name,
Resolution resolution,
String market,
UniverseSettings universeSettings,
PyObject selector
)
```

Creates a new universe and adds it to the algorithm.

Show Details

Parameters		
PyObject	T	The data type.
SecurityType	securityType	The security type the universe produces.
String	name	A unique name for this universe.
Resolution	resolution	The expected resolution of the universe data.
String	market	The market for selected symbols.
UniverseSettings	universeSettings	The subscription settings to use for newly created subscriptions.
PyObject	selector	Function delegate that performs selection on the universe data.

Return

Universe - The new Universe object.

Definition at line 475 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverse() 31/31

```
Universe QuantConnect.Algorithm.QCAlgorithm.AddUniverse (
Type dataType,
SecurityType securityType,
String name,
Resolution resolution,
String market,
UniverseSettings universeSettings,
PyObject pySelector
)
```

Creates a new universe and adds it to the algorithm.

Show Details

Parameters		
Type	dataType	The data type.
SecurityType	securityType	The security type the universe produces.
String	name	A unique name for this universe.
Resolution	resolution	The expected resolution of the universe data.
String	market	The market for selected symbols.
UniverseSettings	universeSettings	The subscription settings to use for newly created subscriptions.
PyObject	pySelector	Function delegate that performs selection on the universe data.

Return

Universe - The new Universe object.

Definition at line 491 of file Algorithm/QCAlgorithm.Python.cs.

UNIVERSES

AddUniverseOptions() 1/3

```
Void QuantConnect.Algorithm.QCAlgorithm.AddUniverseOptions (
Symbol underlyingSymbol,
Func<OptionFilterUniverse, OptionFilterUniverse> optionFilter
)
```

Adds a new universe that creates options of the security by monitoring any changes in the Universe the provided security is in. Additionally, a filter can be applied to the options generated when the universe of the security changes.

Show Details

Parameters		
`Symbol`	underlyingSymbol	Underlying Symbol to add as an option. For Futures, the option chain constructed will be per-contract, as long as a canonical Symbol is provided.
`Func<OptionFilterUniverse, OptionFilterUniverse>`	optionFilter	User-defined filter used to select the options we want out of the option chain provided.

Return

`Void` - This method provides no return.

Definition at line 502 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverseOptions() 2/3

```
Void QuantConnect.Algorithm.QCAlgorithm.AddUniverseOptions (
Universe universe,
Func<OptionFilterUniverse, OptionFilterUniverse> optionFilter
)
```

Creates a new universe selection model and adds it to the algorithm. This universe selection model will chain to the security changes of a given `Universe` selection output and create a new `OptionChainUniverse` for each of them.

Show Details

Parameters		
`Universe`	universe	The universe we want to chain an option universe selection model too.
`Func<OptionFilterUniverse, OptionFilterUniverse>`	optionFilter	The option filter universe to use.

Return

`Void` - This method provides no return.

Definition at line 541 of file Algorithm/QCAlgorithm.Universe.cs.

UNIVERSES

AddUniverseOptions() 3/3

```
Void QuantConnect.Algorithm.QCAlgorithm.AddUniverseOptions (
PyObject universe,
PyObject optionFilter
)
```

Creates a new universe selection model and adds it to the algorithm. This universe selection model will chain to the security changes of a given `Universe` selection output and create a new `OptionChainUniverse` for each of them.

Show Details

Parameters		
`PyObject`	universe	The universe we want to chain an option universe selection model too.
`PyObject`	optionFilter	The option filter universe to use.

Return

`Void` - This method provides no return.

Definition at line 518 of file Algorithm/QCAlgorithm.Python.cs.

ALGORITHM FRAMEWORK UNIVERSES

AddUniverseSelection() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddUniverseSelection (
IUniverseSelectionModel universeSelection
)
```

Adds a new universe selection model.

Show Details

Parameters		
IUniverseSelectionModel	universeSelection	Model defining universes for the algorithm to add.

Return

Void - This method provides no return.

Definition at line 290 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK UNIVERSES

AddUniverseSelection() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.AddUniverseSelection (
PyObject universeSelection
)
```

Adds a new universe selection model.

Show Details

Parameters		
PyObject	universeSelection	Model defining universes for the algorithm to add.

Return

Void - This method provides no return.

Definition at line 121 of file Algorithm/QCAlgorithm.Framework.Python.cs.

INDICATORS

ADDIFF() 1/1

```
AdvanceDeclineDifference QuantConnect.Algorithm.QCAlgorithm.ADDIFF (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, TradeBar>	selector	*(Optional)* /

Return

`AdvanceDeclineDifference` - The new `AdvanceDeclineDifference` object.

Definition at line 2021 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ADOSC() 1/1

```
AccumulationDistributionOscillator QuantConnect.Algorithm.QCAlgorithm.ADOSC (
Symbol symbol,
Int32 fastPeriod,
Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new AccumulationDistributionOscillator indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose ADOSC we want.
Int32	fastPeriod	The fast moving average period.
Int32	slowPeriod	The slow moving average period.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, TradeBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AccumulationDistributionOscillator` - The AccumulationDistributionOscillator indicator for the requested symbol over the specified period.

Definition at line 83 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ADR() 1/1

```
AdvanceDeclineRatio QuantConnect.Algorithm.QCAlgorithm.ADR (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, TradeBar>	selector	*(Optional)* /

Return

`AdvanceDeclineRatio` - The new `AdvanceDeclineRatio` object.

Definition at line 1979 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ADVR() 1/1

```
AdvanceDeclineVolumeRatio QuantConnect.Algorithm.QCAlgorithm.ADVR (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, TradeBar>	selector	*(Optional)* /

Return

`AdvanceDeclineVolumeRatio` - The new `AdvanceDeclineVolumeRatio` object.

Definition at line 2000 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ADX() 1/1

```
AverageDirectionalIndex QuantConnect.Algorithm.QCAlgorithm.ADX (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Average Directional Index indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Average Directional Index we seek.
`Int32`	period	The period over which to compute the Average Directional Index.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AverageDirectionalIndex` - The Average Directional Index indicator for the requested symbol.

Definition at line 124 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ADXR() 1/1

```
AverageDirectionalMovementIndexRating QuantConnect.Algorithm.QCAlgorithm.ADXR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AverageDirectionalMovementIndexRating indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ADXR we want.
`Int32`	period	The period over which to compute the ADXR.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AverageDirectionalMovementIndexRating` - The AverageDirectionalMovementIndexRating indicator for the requested symbol over the specified period.

Definition at line 161 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ALMA() 1/1

```
ArnaudLegouxMovingAverage QuantConnect.Algorithm.QCAlgorithm.ALMA (
Symbol symbol,
Int32 period,
*Int32 sigma,
*Decimal offset,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new ArnaudLegouxMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ALMA we want.
`Int32`	period	int - the number of periods to calculate the ALMA.
`*Int32`	sigma	*(Optional)* int - this parameter is responsible for the shape of the curve coefficients. .
`*Decimal`	offset	*(Optional)* decimal - This parameter allows regulating the smoothness and high sensitivity of the Moving Average. The range for this parameter is [0, 1]. .
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ArnaudLegouxMovingAverage` - The ArnaudLegouxMovingAverage indicator for the requested symbol over the specified period.

Definition at line 185 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

AO() 1/1

```
AwesomeOscillator QuantConnect.Algorithm.QCAlgorithm.AO (
Symbol symbol,
Int32 slowPeriod,
Int32 fastPeriod,
MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Awesome Oscillator from the specified periods.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Awesome Oscillator we seek.
`Int32`	slowPeriod	The period of the slow moving average associated with the AO.
`Int32`	fastPeriod	The period of the fast moving average associated with the AO.
`MovingAverageType`	type	The type of moving average used when computing the fast and slow term. Defaults to simple moving average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AwesomeOscillator` - The new `AwesomeOscillator` object.

Definition at line 143 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

APO() 1/1

```
AbsolutePriceOscillator QuantConnect.Algorithm.QCAlgorithm.APO (
Symbol symbol,
Int32 fastPeriod,
Int32 slowPeriod,
MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new AbsolutePriceOscillator indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose APO we want.
Int32	fastPeriod	The fast moving average period.
Int32	slowPeriod	The slow moving average period.
MovingAverageType	movingAverageType	The type of moving average to use.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AbsolutePriceOscillator` - The AbsolutePriceOscillator indicator for the requested symbol over the specified period.

Definition at line 205 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

APS() 1/1

```
AugenPriceSpike QuantConnect.Algorithm.QCAlgorithm.APS (
 Symbol symbol,
 *Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates an AugenPriceSpike indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose APS we want.
`*Int32`	period	*(Optional)* The period of the APS.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`AugenPriceSpike` - The AugenPriceSpike indicator for the given parameters.

Definition at line 277 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ARIMA() 1/1

```
AutoRegressiveIntegratedMovingAverage QuantConnect.Algorithm.QCAlgorithm.ARIMA (
Symbol symbol,
Int32 arOrder,
Int32 diffOrder,
Int32 maOrder,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new ARIMA indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose ARIMA indicator we want.
Int32	arOrder	AR order (p) -- defines the number of past values to consider in the AR component of the model.
Int32	diffOrder	Difference order (d) -- defines how many times to difference the model before fitting parameters.
Int32	maOrder	MA order (q) -- defines the number of past values to consider in the MA component of the model.
Int32	period	Size of the rolling series to fit onto.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

AutoRegressiveIntegratedMovingAverage - The ARIMA indicator for the requested symbol over the specified period.

Definition at line 104 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

AROON() 1/2

```
AroonOscillator QuantConnect.Algorithm.QCAlgorithm.AROON (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AroonOscillator indicator which will compute the AroonUp and AroonDown (as well as the delta).

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Aroon we seek.
`Int32`	period	The look back period for computing number of periods since maximum and minimum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AroonOscillator` - An AroonOscillator configured with the specified periods.

Definition at line 223 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

AROON() 2/2

```
AroonOscillator QuantConnect.Algorithm.QCAlgorithm.AROON (
 Symbol symbol,
 Int32 upPeriod,
 Int32 downPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AroonOscillator indicator which will compute the AroonUp and AroonDown (as well as the delta).

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Aroon we seek.
`Int32`	upPeriod	The look back period for computing number of periods since maximum.
`Int32`	downPeriod	The look back period for computing number of periods since minimum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AroonOscillator` - An AroonOscillator configured with the specified periods.

Definition at line 238 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ASI() 1/1

```
WilderAccumulativeSwingIndex QuantConnect.Algorithm.QCAlgorithm.ASI (
 Symbol symbol,
 Decimal limitMove,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a Wilder Accumulative Swing Index (ASI) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ASI we want.
`Decimal`	limitMove	The maximum daily change in price for the ASI.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`WilderAccumulativeSwingIndex` - The WilderAccumulativeSwingIndex for the given parameters.

Definition at line 1927 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ATR() 1/1

```
AverageTrueRange QuantConnect.Algorithm.QCAlgorithm.ATR (
Symbol symbol,
Int32 period,
*MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new AverageTrueRange indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose ATR we want.
Int32	period	The smoothing period used to smooth the computed TrueRange values.
*MovingAverageType	type	*(Optional)* The type of smoothing to use.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, IBaseDataBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`AverageTrueRange` - A new AverageTrueRange indicator with the specified smoothing type and period.

Definition at line 258 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

B() 1/1

```
Beta QuantConnect.Algorithm.QCAlgorithm.B (
 Symbol target,
 Symbol reference,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a Beta indicator for the given target symbol in relation with the reference used. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	target	The target symbol whose Beta value we want.
`Symbol`	reference	The reference symbol to compare with the target symbol.
`Int32`	period	The period of the Beta indicator.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`Beta` - The Beta indicator for the given parameters.

Definition at line 318 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

BB() 1/1

```
BollingerBands QuantConnect.Algorithm.QCAlgorithm.BB (
 Symbol symbol,
 Int32 period,
 Decimal k,
 *MovingAverageType movingAverageType,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
 )
```

Creates a new BollingerBands indicator which will compute the MiddleBand, UpperBand, LowerBand, and StandardDeviation.

Show Details

Parameters		
Symbol	symbol	The symbol whose BollingerBands we seek.
Int32	period	The period of the standard deviation and moving average (middle band).
Decimal	k	The number of standard deviations specifying the distance between the middle band and upper or lower bands.
*MovingAverageType	movingAverageType	*(Optional)* The type of moving average to be used.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

BollingerBands - A BollingerBands configured with the specified period.

Definition at line 297 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

BOP() 1/1

```
BalanceOfPower QuantConnect.Algorithm.QCAlgorithm.BOP (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Balance Of Power indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Balance Of Power we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`BalanceOfPower` - The Balance Of Power indicator for the requested symbol.

Definition at line 337 of file Algorithm/QCAlgorithm.Indicators.cs.

TRADING AND ORDERS

Buy() 1/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Buy (
 Symbol symbol,
 Decimal quantity
)
```

Buy Stock (Alias of Order).

Show Details

Parameters		
`Symbol`	symbol	string Symbol of the asset to trade.
`Decimal`	quantity	decimal Quantity of the asset to trade.

Return

`OrderTicket` - The order ticket instance.

Definition at line 75 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Buy() 2/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Buy (
Symbol symbol,
Single quantity
)
```

Buy Stock (Alias of Order).

Show Details

Parameters		
Symbol	symbol	string Symbol of the asset to trade.
Single	quantity	float Quantity of the asset to trade.

Return

`OrderTicket` - The order ticket instance.

Definition at line 88 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Buy() 3/5

```
IEnumerable<OrderTicket> QuantConnect.Algorithm.QCAlgorithm.Buy (
OptionStrategy strategy,
Int32 quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Buy Option Strategy (Alias of Order).

Show Details

Parameters		
`OptionStrategy`	strategy	Specification of the strategy to trade.
`Int32`	quantity	Quantity of the strategy to trade.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`IEnumerable<OrderTicket>` - Sequence of order tickets.

Definition at line 628 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Buy() 4/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Buy (
 Symbol symbol,
 Int32 quantity
)
```

Buy Stock (Alias of Order).

Show Details

Parameters		
`Symbol`	symbol	string Symbol of the asset to trade.
`Int32`	quantity	int Quantity of the asset to trade.

Return

`OrderTicket` - The order ticket instance.

Definition at line 49 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Buy() 5/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Buy (
 Symbol symbol,
 Double quantity
)
```

Buy Stock (Alias of Order).

Show Details

Parameters		
Symbol	symbol	string Symbol of the asset to trade.
Double	quantity	double Quantity of the asset to trade.

Return

OrderTicket - The order ticket instance.

Definition at line 62 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

CalculateOrderQuantity() 1/2

```
Decimal QuantConnect.Algorithm.QCAlgorithm.CalculateOrderQuantity (
 Symbol symbol,
 Double target
)
```

Calculate the order quantity to achieve target-percent holdings.

Show Details

Parameters		
Symbol	symbol	Security object we're asking for.
Double	target	Target percentage holdings.

Return

Decimal - Order quantity to achieve this percentage.

Definition at line 1304 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

CalculateOrderQuantity() 2/2

```
Decimal QuantConnect.Algorithm.QCAlgorithm.CalculateOrderQuantity (
 Symbol symbol,
 Decimal target
)
```

Calculate the order quantity to achieve target-percent holdings.

Show Details

Parameters		
Symbol	symbol	Security object we're asking for.
Decimal	target	Target percentage holdings, this is an unleveraged value, so if you have 2x leverage and request 100% holdings, it will utilize half of the available margin.

Return

Decimal - Order quantity to achieve this percentage.

Definition at line 1318 of file Algorithm/QCAlgorithm.Trading.cs.

INDICATORS

CC() 1/1

```
CoppockCurve QuantConnect.Algorithm.QCAlgorithm.CC (
Symbol symbol,
*Int32 shortRocPeriod,
*Int32 longRocPeriod,
*Int32 lwmaPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Initializes a new instance of the CoppockCurve" indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Coppock Curve we want.
`*Int32`	shortRocPeriod	*(Optional)* The period for the short ROC.
`*Int32`	longRocPeriod	*(Optional)* The period for the long ROC.
`*Int32`	lwmaPeriod	*(Optional)* The period for the LWMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`CoppockCurve` - The Coppock Curve indicator for the requested symbol over the specified period.

Definition at line 357 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

CCI() 1/1

```
CommodityChannelIndex QuantConnect.Algorithm.QCAlgorithm.CCI (
Symbol symbol,
Int32 period,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new CommodityChannelIndex indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose CCI we want.
`Int32`	period	The period over which to compute the CCI.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use in computing the typical price average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`CommodityChannelIndex` - The CommodityChannelIndex indicator for the requested symbol over the specified period.

Definition at line 378 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

CMF() 1/1

```
ChaikinMoneyFlow QuantConnect.Algorithm.QCAlgorithm.CMF (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new ChaikinMoneyFlow indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose CMF we want.
`Int32`	period	The period over which to compute the CMF.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`ChaikinMoneyFlow` - The ChaikinMoneyFlow indicator for the requested symbol over the specified period.

Definition at line 396 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

CMO() 1/1

```
ChandeMomentumOscillator QuantConnect.Algorithm.QCAlgorithm.CMO (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new ChandeMomentumOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose CMO we want.
`Int32`	period	The period over which to compute the CMO.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ChandeMomentumOscillator` - The ChandeMomentumOscillator indicator for the requested symbol over the specified period.

Definition at line 415 of file Algorithm/QCAlgorithm.Indicators.cs.

<-- Missing documentation attribute for ComboLegLimitOrder -->

<-- Missing documentation attribute for ComboLimitOrder -->

<-- Missing documentation attribute for ComboMarketOrder -->

HANDLING DATA SECURITIES AND PORTFOLIO

CompositeFIGI() 1/2

```
Symbol QuantConnect.Algorithm.QCAlgorithm.CompositeFIGI (
String compositeFigi,
*Nullable<DateTime> tradingDate
)
```

Converts a composite FIGI identifier into a `Symbol` .

Show Details

Parameters		
`String`	compositeFigi	The composite Financial Instrument Global Identifier (FIGI) of an asset.
`*Nullable<DateTime>`	tradingDate	*(Optional)* The date that the stock being looked up is/was traded at. The date is used to create a Symbol with the ticker set to the ticker the asset traded under on the trading date. .

Return

`Symbol` - Symbol corresponding to the composite FIGI. If no Symbol with a matching composite FIGI was found, returns None.

Definition at line 2928 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA SECURITIES AND PORTFOLIO

CompositeFIGI() 2/2

```
String QuantConnect.Algorithm.QCAlgorithm.CompositeFIGI (
Symbol symbol
)
```

Converts a `Symbol` into a composite FIGI identifier.

Show Details

Parameters		
`Symbol`	symbol	The `Symbol` .

Return

`String` - Composite FIGI corresponding to the Symbol. If no matching composite FIGI is found, returns None.

Definition at line 2940 of file Algorithm/QCAlgorithm.cs.

CONSOLIDATING DATA

Consolidate() 1/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Resolution period,
PyObject handler
)
```

Registers the `handler` to receive consolidated data for the specified symbol.

Show Details

Parameters		
`Symbol`	symbol	The symbol who's data is to be consolidated.
`Resolution`	period	The consolidation period.
`PyObject`	handler	Data handler receives new consolidated data when generated.

Return

`IDataConsolidator` - A new consolidator matching the requested parameters with the handler already registered.

Definition at line 1257 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA

Consolidate() 2/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Resolution period,
Nullable<TickType> tickType,
PyObject handler
)
```

Registers the `handler` to receive consolidated data for the specified symbol.

Show Details

Parameters		
`Symbol`	symbol	The symbol who's data is to be consolidated.
`Resolution`	period	The consolidation period.
`Nullable<TickType>`	tickType	The tick type of subscription used as data source for consolidator. Specify None to use first subscription found.
`PyObject`	handler	Data handler receives new consolidated data when generated.

Return

`IDataConsolidator` - A new consolidator matching the requested parameters with the handler already registered.

Definition at line 1271 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA

Consolidate() 3/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
TimeSpan period,
PyObject handler
)
```

Registers the `handler` to receive consolidated data for the specified symbol.

Show Details

Parameters		
`Symbol`	symbol	The symbol who's data is to be consolidated.
`TimeSpan`	period	The consolidation period.
`PyObject`	handler	Data handler receives new consolidated data when generated.

Return

`IDataConsolidator` - A new consolidator matching the requested parameters with the handler already registered.

Definition at line 1284 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA

Consolidate() 4/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
TimeSpan period,
Nullable<TickType> tickType,
PyObject handler
)
```

Registers the `handler` to receive consolidated data for the specified symbol.

Show Details

Parameters		
Symbol	symbol	The symbol who's data is to be consolidated.
TimeSpan	period	The consolidation period.
Nullable<TickType>	tickType	The tick type of subscription used as data source for consolidator. Specify None to use first subscription found.
PyObject	handler	Data handler receives new consolidated data when generated.

Return

IDataConsolidator - A new consolidator matching the requested parameters with the handler already registered.

Definition at line 1298 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA

Consolidate() 5/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Func<DateTime, CalendarInfo> calendar,
PyObject handler
)
```

Registers the handler to receive consolidated data for the specified symbol.

Show Details

Parameters		
Symbol	symbol	The symbol who's data is to be consolidated.
Func<DateTime, CalendarInfo>	calendar	The consolidation calendar.
PyObject	handler	Data handler receives new consolidated data when generated.

Return

IDataConsolidator - A new consolidator matching the requested parameters with the handler already registered.

Definition at line 1324 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA

Consolidate() 6/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Func<DateTime, CalendarInfo> calendar,
Nullable<TickType> tickType,
PyObject handler
)
```

Registers the `handler` to receive consolidated data for the specified symbol.

Show Details

Parameters		
`Symbol`	symbol	The symbol who's data is to be consolidated.
`Func<DateTime, CalendarInfo>`	calendar	The consolidation calendar.
`Nullable<TickType>`	tickType	The tick type of subscription used as data source for consolidator. Specify None to use first subscription found.
`PyObject`	handler	Data handler receives new consolidated data when generated.

Return

`IDataConsolidator` - A new consolidator matching the requested parameters with the handler already registered.

Definition at line 1362 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA

Consolidate() 7/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Resolution period,
Action<TradeBar> handler
)
```

Show Details

Parameters		
Symbol	symbol	/
Resolution	period	/
Action<TradeBar>	handler	/

Return

IDataConsolidator - The new IDataConsolidator object.

Definition at line 2684 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 8/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
 Symbol symbol,
 TimeSpan period,
 Action<TradeBar> handler
 )
```

Show Details

Parameters		
Symbol	symbol	/
TimeSpan	period	/
Action<TradeBar>	handler	/

Return

IDataConsolidator - The new IDataConsolidator object.

Definition at line 2697 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 9/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Resolution period,
Action<QuoteBar> handler
)
```

Show Details

Parameters		
Symbol	symbol	/
Resolution	period	/
Action<QuoteBar>	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2710 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 10/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
TimeSpan period,
Action<QuoteBar> handler
)
```

Show Details

Parameters		
Symbol	symbol	/
TimeSpan	period	/
Action<QuoteBar>	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2723 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 11/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
TimeSpan period,
Action<T> handler
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`TimeSpan`	period	/
`Action<T>`	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2737 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 12/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Resolution period,
Nullable<TickType> tickType,
Action<T> handler
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`Resolution`	period	/
`Nullable<TickType>`	tickType	/
`Action<T>`	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2757 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 13/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
 Symbol symbol,
 TimeSpan period,
 Nullable<TickType> tickType,
 Action<T> handler
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`TimeSpan`	period	/
`Nullable<TickType>`	tickType	/
`Action<T>`	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2773 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 14/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Func<DateTime, CalendarInfo> calendar,
Action<QuoteBar> handler
)
```

Show Details

Parameters		
Symbol	symbol	/
Func<DateTime, CalendarInfo>	calendar	/
Action<QuoteBar>	handler	/

Return

IDataConsolidator - The new IDataConsolidator object.

Definition at line 2794 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 15/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Func<DateTime, CalendarInfo> calendar,
Action<TradeBar> handler
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`Func<DateTime, CalendarInfo>`	calendar	/
`Action<TradeBar>`	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2807 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 16/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Func<DateTime, CalendarInfo> calendar,
Action<T> handler
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`Func<DateTime, CalendarInfo>`	calendar	/
`Action<T>`	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2821 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

Consolidate() 17/17

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.Consolidate (
Symbol symbol,
Func<DateTime, CalendarInfo> calendar,
Nullable<TickType> tickType,
Action<T> handler
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`Func<DateTime, CalendarInfo>`	calendar	/
`Nullable<TickType>`	tickType	/
`Action<T>`	handler	/

Return

`IDataConsolidator` - The new `IDataConsolidator` object.

Definition at line 2841 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA

CreateConsolidator() 1/1

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.CreateConsolidator (
TimeSpan period,
Type consolidatorInputType,
*Nullable<TickType> tickType
)
```

Creates a new consolidator for the specified period, generating the requested output type.

Show Details

Parameters		
TimeSpan	period	The consolidation period.
Type	consolidatorInputType	The desired input type of the consolidator, such as TradeBar or QuoteBar.
*Nullable<TickType>	tickType	*(Optional)* Trade or Quote. Optional, defaults to trade.

Return

IDataConsolidator - A new consolidator matching the requested parameters.

Definition at line 2607 of file Algorithm/QCAlgorithm.Indicators.cs.

HANDLING DATA SECURITIES AND PORTFOLIO

CUSIP() 1/2

```
Symbol QuantConnect.Algorithm.QCAlgorithm.CUSIP (
 String cusip,
 *Nullable<DateTime> tradingDate
)
```

Converts a CUSIP identifier into a Symbol .

Show Details

Parameters		
String	cusip	The CUSIP number of an asset.
*Nullable<DateTime>	tradingDate	*(Optional)* The date that the stock being looked up is/was traded at. The date is used to create a Symbol with the ticker set to the ticker the asset traded under on the trading date. .

Return

Symbol - Symbol corresponding to the CUSIP. If no Symbol with a matching CUSIP was found, returns None.

Definition at line 2956 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA SECURITIES AND PORTFOLIO

CUSIP() 2/2

```
String QuantConnect.Algorithm.QCAlgorithm.CUSIP (
Symbol symbol
)
```

Converts a `Symbol` into a CUSIP identifier.

Show Details

Parameters		
`Symbol`	symbol	The `Symbol` .

Return

`String` - CUSIP corresponding to the Symbol. If no matching CUSIP is found, returns None.

Definition at line 2968 of file Algorithm/QCAlgorithm.cs.

INDICATORS

DCH() 1/2

```
DonchianChannel QuantConnect.Algorithm.QCAlgorithm.DCH (
Symbol symbol,
Int32 upperPeriod,
Int32 lowerPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Donchian Channel indicator which will compute the Upper Band and Lower Band. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose Donchian Channel we seek.
Int32	upperPeriod	The period over which to compute the upper Donchian Channel.
Int32	lowerPeriod	The period over which to compute the lower Donchian Channel.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, IBaseDataBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

DonchianChannel - The Donchian Channel indicator for the requested symbol.

Definition at line 454 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

DCH() 2/2

```
DonchianChannel QuantConnect.Algorithm.QCAlgorithm.DCH (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Overload shorthand to create a new symmetric Donchian Channel indicator which has the upper and lower channels set to the same period length.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Donchian Channel we seek.
`Int32`	period	The period over which to compute the Donchian Channel.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`DonchianChannel` - The Donchian Channel indicator for the requested symbol.

Definition at line 473 of file Algorithm/QCAlgorithm.Indicators.cs.

LOGGING

Debug() 1/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Debug (
 PyObject message
)
```

Send a debug message to the web console:.

Show Details

Parameters		
`PyObject`	message	Message to send to debug console.

Return

`Void` - This method provides no return.

Definition at line 1210 of file Algorithm/QCAlgorithm.Python.cs.

LOGGING

Debug() 2/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Debug (
String message
)
```

Send a debug message to the web console:.

Show Details

Parameters		
String	message	Message to send to debug console.

Return

Void - This method provides no return.

Definition at line 2453 of file Algorithm/QCAlgorithm.cs.

LOGGING

Debug() 3/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Debug (
Int32 message
)
```

Send a debug message to the web console:.

Show Details

Parameters		
Int32	message	Message to send to debug console.

Return

Void - This method provides no return.

Definition at line 2467 of file Algorithm/QCAlgorithm.cs.

LOGGING

Debug() 4/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Debug (
Double message
)
```

Send a debug message to the web console:.

Show Details

Parameters		
Double	message	Message to send to debug console.

Return

Void - This method provides no return.

Definition at line 2479 of file Algorithm/QCAlgorithm.cs.

LOGGING

Debug() 5/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Debug (
Decimal message
)
```

Send a debug message to the web console:.

Show Details

Parameters		
Decimal	message	Message to send to debug console.

Return

Void - This method provides no return.

Definition at line 2491 of file Algorithm/QCAlgorithm.cs.

INDICATORS

DEM() 1/1

```
DeMarkerIndicator QuantConnect.Algorithm.QCAlgorithm.DEM (
Symbol symbol,
Int32 period,
MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new DeMarker Indicator (DEM), an oscillator-type indicator measuring changes in terms of an asset's High and Low tradebar values.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose DEM we seek.
`Int32`	period	The period of the moving average implemented.
`MovingAverageType`	type	Specifies the type of moving average to be used.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`DeMarkerIndicator` - The DeMarker indicator for the requested symbol.

Definition at line 435 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

DEMA() 1/1

```
DoubleExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.DEMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new DoubleExponentialMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose DEMA we want.
`Int32`	period	The period over which to compute the DEMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`DoubleExponentialMovingAverage` - The DoubleExponentialMovingAverage indicator for the requested symbol over the specified period.

Definition at line 487 of file Algorithm/QCAlgorithm.Indicators.cs.

ADDING DATA | MACHINE LEARNING

Download() 1/5

```
String QuantConnect.Algorithm.QCAlgorithm.Download (
String address,
PyObject headers
)
```

Downloads the requested resource as a `String` . The resource to download is specified as a `String` containing the URI.

Show Details

Parameters		
String	address	A string containing the URI to download.
PyObject	headers	Defines header values to add to the request.

Return

String - The requested resource as a.

Definition at line 1160 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA MACHINE LEARNING

Download() 2/5

```
String QuantConnect.Algorithm.QCAlgorithm.Download (
 String address,
 PyObject headers,
 String userName,
 String password
)
```

Downloads the requested resource as a String . The resource to download is specified as a String containing the URI.

Show Details

Parameters		
String	address	A string containing the URI to download.
PyObject	headers	Defines header values to add to the request.
String	userName	The user name associated with the credentials.
String	password	The password for the user name associated with the credentials.

Return

String - The requested resource as a.

Definition at line 1173 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA MACHINE LEARNING

Download() 3/5

```
String QuantConnect.Algorithm.QCAlgorithm.Download (
 String address
)
```

Downloads the requested resource as a `String` . The resource to download is specified as a `String` containing the URI.

Show Details

Parameters		
`String`	address	A string containing the URI to download.

Return

`String` - The requested resource as a.

Definition at line 2729 of file Algorithm/QCAlgorithm.cs.

ADDING DATA MACHINE LEARNING

Download() 4/5

```
String QuantConnect.Algorithm.QCAlgorithm.Download (
 String address,
 IEnumerable<KeyValuePair<String, String>> headers
)
```

Show Details

Parameters		
`String`	address	/
`IEnumerable<KeyValuePair<String, String>>`	headers	/

Return

`String` - The new `String` object.

Definition at line 2740 of file Algorithm/QCAlgorithm.cs.

ADDING DATA MACHINE LEARNING

Download() 5/5

```
String QuantConnect.Algorithm.QCAlgorithm.Download (
String address,
IEnumerable<KeyValuePair<String, String>> headers,
String userName,
String password
)
```

Show Details

Parameters		
`String`	address	/
`IEnumerable<KeyValuePair<String, String>>`	headers	/
`String`	userName	/
`String`	password	/

Return

`String` - The new `String` object.

Definition at line 2753 of file Algorithm/QCAlgorithm.cs.

INDICATORS

DPO() 1/1

```
DetrendedPriceOscillator QuantConnect.Algorithm.QCAlgorithm.DPO (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new DetrendedPriceOscillator" indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose DPO we want.
`Int32`	period	The period over which to compute the DPO.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`DetrendedPriceOscillator` - A new registered DetrendedPriceOscillator indicator for the requested symbol over the specified period.

Definition at line 505 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

EMA() 1/2

```
ExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.EMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an ExponentialMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EMA we want.
`Int32`	period	The period of the EMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ExponentialMovingAverage` - The ExponentialMovingAverage for the given parameters.

Definition at line 524 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

EMA() 2/2

```
ExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.EMA (
 Symbol symbol,
 Int32 period,
 Decimal smoothingFactor,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates an ExponentialMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EMA we want.
`Int32`	period	The period of the EMA.
`Decimal`	smoothingFactor	The percentage of data from the previous value to be carried into the next value.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`ExponentialMovingAverage` - The ExponentialMovingAverage for the given parameters.

Definition at line 540 of file Algorithm/QCAlgorithm.Indicators.cs.

ALGORITHM FRAMEWORK

EmitInsights() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.EmitInsights (
Insight> insights
)
```

Manually emit insights from an algorithm. This is typically invoked before calls to submit orders in algorithms written against QCAlgorithm that have been ported into the algorithm framework.

Show Details

Parameters		
`Insight>`	insights	The array of insights to be emitted.

Return

`Void` - This method provides no return.

Definition at line 412 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK

EmitInsights() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.EmitInsights (
Insight insight
)
```

Manually emit insights from an algorithm. This is typically invoked before calls to submit orders in algorithms written against QCAlgorithm that have been ported into the algorithm framework.

Show Details

Parameters		
Insight	insight	The insight to be emitted.

Return

Void - This method provides no return.

Definition at line 436 of file Algorithm/QCAlgorithm.Framework.cs.

INDICATORS

EMV() 1/1

```
EaseOfMovementValue QuantConnect.Algorithm.QCAlgorithm.EMV (
Symbol symbol,
*Int32 period,
*Int32 scale,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an EaseOfMovementValue indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EMV we want.
`*Int32`	period	*(Optional)* The period of the EMV.
`*Int32`	scale	*(Optional)* The length of the outputed value.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`EaseOfMovementValue` - The EaseOfMovementValue indicator for the given parameters.

Definition at line 560 of file Algorithm/QCAlgorithm.Indicators.cs.

LOGGING

Error() 1/6

```
Void QuantConnect.Algorithm.QCAlgorithm.Error (
 PyObject message
)
```

Send a string error message to the Console.

Show Details

Parameters		
`PyObject`	message	Message to display in errors grid.

Return

`Void` - This method provides no return.

Definition at line 1222 of file Algorithm/QCAlgorithm.Python.cs.

LOGGING

Error() 2/6

```
Void QuantConnect.Algorithm.QCAlgorithm.Error (
 String message
)
```

Send a string error message to the Console.

Show Details

Parameters		
String	message	Message to display in errors grid.

Return

Void - This method provides no return.

Definition at line 2552 of file Algorithm/QCAlgorithm.cs.

LOGGING

Error() 3/6

```
Void QuantConnect.Algorithm.QCAlgorithm.Error (
 Int32 message
)
```

Send a int error message to the Console.

Show Details

Parameters		
Int32	message	Message to display in errors grid.

Return

Void - This method provides no return.

Definition at line 2566 of file Algorithm/QCAlgorithm.cs.

LOGGING

Error() 4/6

```
Void QuantConnect.Algorithm.QCAlgorithm.Error (
Double message
)
```

Send a double error message to the Console.

Show Details

Parameters		
Double	message	Message to display in errors grid.

Return

Void - This method provides no return.

Definition at line 2578 of file Algorithm/QCAlgorithm.cs.

LOGGING

Error() 5/6

```
Void QuantConnect.Algorithm.QCAlgorithm.Error (
Decimal message
)
```

Send a decimal error message to the Console.

Show Details

Parameters		
Decimal	message	Message to display in errors grid.

Return

`Void` - This method provides no return.

Definition at line 2590 of file Algorithm/QCAlgorithm.cs.

LOGGING

Error() 6/6

```
Void QuantConnect.Algorithm.QCAlgorithm.Error (
Exception error
)
```

Send a string error message to the Console.

Show Details

Parameters		
`Exception`	error	Exception object captured from a try catch loop.

Return

`Void` - This method provides no return.

Definition at line 2602 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

ExerciseOption() 1/1

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.ExerciseOption (
Symbol optionSymbol,
Int32 quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Send an exercise order to the transaction handler.

Show Details

Parameters		
`Symbol`	optionSymbol	String symbol for the option position.
`Int32`	quantity	Quantity of options contracts.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 589 of file Algorithm/QCAlgorithm.Trading.cs.

INDICATORS

FilteredIdentity() 1/6

```
FilteredIdentity QuantConnect.Algorithm.QCAlgorithm.FilteredIdentity (
 Symbol symbol,
 *PyObject selector,
 *PyObject filter,
 *String fieldName
)
```

Creates a new FilteredIdentity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`*PyObject`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*PyObject`	filter	*(Optional)* Filters the IBaseData send into the indicator, if None defaults to True (x => True) which means no filter.
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`FilteredIdentity` - A new FilteredIdentity indicator for the specified symbol and selector.

Definition at line 791 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS

FilteredIdentity() 2/6

```
FilteredIdentity QuantConnect.Algorithm.QCAlgorithm.FilteredIdentity (
 Symbol symbol,
 Resolution resolution,
 *PyObject selector,
 *PyObject filter,
 *String fieldName
)
```

Creates a new FilteredIdentity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`Resolution`	resolution	The desired resolution of the data.
`*PyObject`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*PyObject`	filter	*(Optional)* Filters the IBaseData send into the indicator, if None defaults to True (x => True) which means no filter.
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`FilteredIdentity` - A new FilteredIdentity indicator for the specified symbol and selector.

Definition at line 808 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS

FilteredIdentity() 3/6

```
FilteredIdentity QuantConnect.Algorithm.QCAlgorithm.FilteredIdentity (
 Symbol symbol,
 TimeSpan resolution,
 *PyObject selector,
 *PyObject filter,
 *String fieldName
)
```

Creates a new FilteredIdentity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose values we want as an indicator.
TimeSpan	resolution	The desired resolution of the data.
*PyObject	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
*PyObject	filter	*(Optional)* Filters the IBaseData send into the indicator, if None defaults to True (x => True) which means no filter.
*String	fieldName	*(Optional)* The name of the field being selected.

Return

FilteredIdentity - A new FilteredIdentity indicator for the specified symbol and selector.

Definition at line 829 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS

FilteredIdentity() 4/6

```
FilteredIdentity QuantConnect.Algorithm.QCAlgorithm.FilteredIdentity (
 Symbol symbol,
 *Func<IBaseData, IBaseDataBar> selector,
 *Func<IBaseData, Boolean> filter,
 *String fieldName
)
```

Creates a new FilteredIdentity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.
`*Func<IBaseData, Boolean>`	filter	*(Optional)* Filters the IBaseData send into the indicator, if None defaults to True (x => True) which means no filter.
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`FilteredIdentity` - A new FilteredIdentity indicator for the specified symbol and selector.

Definition at line 579 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

FilteredIdentity() 5/6

```
FilteredIdentity QuantConnect.Algorithm.QCAlgorithm.FilteredIdentity (
 Symbol symbol,
 Resolution resolution,
 *Func<IBaseData, IBaseDataBar> selector,
 *Func<IBaseData, Boolean> filter,
 *String fieldName
)
```

Creates a new FilteredIdentity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`Resolution`	resolution	The desired resolution of the data.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.
`*Func<IBaseData, Boolean>`	filter	*(Optional)* Filters the IBaseData send into the indicator, if None defaults to True (x => True) which means no filter.
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`FilteredIdentity` - A new FilteredIdentity indicator for the specified symbol and selector.

Definition at line 596 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

FilteredIdentity() 6/6

```
FilteredIdentity QuantConnect.Algorithm.QCAlgorithm.FilteredIdentity (
 Symbol symbol,
 TimeSpan resolution,
 *Func<IBaseData, IBaseDataBar> selector,
 *Func<IBaseData, Boolean> filter,
 *String fieldName
)
```

Creates a new FilteredIdentity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose values we want as an indicator.
TimeSpan	resolution	The desired resolution of the data.
*Func<IBaseData, IBaseDataBar>	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
*Func<IBaseData, Boolean>	filter	*(Optional)* Filters the IBaseData send into the indicator, if None defaults to True (x => True) which means no filter.
*String	fieldName	*(Optional)* The name of the field being selected.

Return

`FilteredIdentity` - A new FilteredIdentity indicator for the specified symbol and selector.

Definition at line 615 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

FISH() 1/1

```
FisherTransform QuantConnect.Algorithm.QCAlgorithm.FISH (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates an FisherTransform indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose FisherTransform we want.
`Int32`	period	The period of the FisherTransform.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`FisherTransform` - The FisherTransform for the given parameters.

Definition at line 633 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

FRAMA() 1/1

```
FractalAdaptiveMovingAverage QuantConnect.Algorithm.QCAlgorithm.FRAMA (
Symbol symbol,
Int32 period,
*Int32 longPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates an FractalAdaptiveMovingAverage (FRAMA) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose FRAMA we want.
`Int32`	period	The period of the FRAMA.
`*Int32`	longPeriod	*(Optional)* The long period of the FRAMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`FractalAdaptiveMovingAverage` - The FRAMA for the given parameters.

Definition at line 653 of file Algorithm/QCAlgorithm.Indicators.cs.

ALGORITHM FRAMEWORK

FrameworkPostInitialize() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.FrameworkPostInitialize (
)
```

Called by setup handlers after Initialize and allows the algorithm a chance to organize the data gather in the Initialize method.

Show Details

This method requires no argument input.

Return

`Void` - This method provides no return.

Definition at line 84 of file Algorithm/QCAlgorithm.Framework.cs.

ADDING DATA HISTORICAL DATA

GetLastKnownPrices() 1/2

```
IEnumerable<BaseData> QuantConnect.Algorithm.QCAlgorithm.GetLastKnownPrices (
Security security
)
```

Yields data to warmup a security for all it's subscribed data types.

Show Details

Parameters		
Security	security	Security object for which to retrieve historical data.

Return

IEnumerable<BaseData> - Securities historical data.

Definition at line 629 of file Algorithm/QCAlgorithm.History.cs.

ADDING DATA HISTORICAL DATA

GetLastKnownPrices() 2/2

```
IEnumerable<BaseData> QuantConnect.Algorithm.QCAlgorithm.GetLastKnownPrices (
Symbol symbol
)
```

Yields data to warmup a security for all it's subscribed data types.

Show Details

Parameters		
Symbol	symbol	The symbol we want to get seed data for.

Return

IEnumerable<BaseData> - Securities historical data.

Definition at line 641 of file Algorithm/QCAlgorithm.History.cs.

PARAMETER AND OPTIMIZATION

GetParameters() 1/1

```
IReadOnlyDictionary<String, String> QuantConnect.Algorithm.QCAlgorithm.GetParameters (

)
```

Gets a read-only dictionary with all current parameters.

Show Details

This method requires no argument input.

Return

`IReadOnlyDictionary<String, String>` - The new `IReadOnlyDictionary<String, String>` object.

Definition at line 730 of file Algorithm/QCAlgorithm.cs.

INDICATORS

HeikinAshi() 1/1

```
HeikinAshi QuantConnect.Algorithm.QCAlgorithm.HeikinAshi (
 Symbol symbol,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new Heikin-Ashi indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Heikin-Ashi we want.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`HeikinAshi` - The Heikin-Ashi indicator for the requested symbol over the specified period.

Definition at line 670 of file Algorithm/QCAlgorithm.Indicators.cs.

HISTORICAL DATA

History() 1/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
 PyObject tickers,
 Int32 periods,
 *Nullable<Resolution> resolution,
 *Nullable<Boolean> fillForward,
 *Nullable<Boolean> extendedMarketHours,
 *Nullable<DataMappingMode> dataMappingMode,
 *Nullable<DataNormalizationMode> dataNormalizationMode,
 *Nullable<Int32> contractDepthOffset
 )
```

Gets the historical data for the specified symbol. The exact number of bars will be returned. The symbol must exist in the Securities collection.

Show Details

Parameters		
PyObject	tickers	The symbols to retrieve historical data for.
Int32	periods	The number of bars to request.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - A python dictionary with pandas DataFrame containing the requested historical data.

Definition at line 854 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 2/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
 PyObject tickers,
 TimeSpan span,
 *Nullable<Resolution> resolution,
 *Nullable<Boolean> fillForward,
 *Nullable<Boolean> extendedMarketHours,
 *Nullable<DataMappingMode> dataMappingMode,
 *Nullable<DataNormalizationMode> dataNormalizationMode,
 *Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbols over the requested span. The symbols must exist in the Securities collection.

Show Details

Parameters		
PyObject	tickers	The symbols to retrieve historical data for.
TimeSpan	span	The span over which to retrieve recent historical data.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - A python dictionary with pandas DataFrame containing the requested historical data.

Definition at line 878 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 3/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
PyObject tickers,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbols between the specified dates. The symbols must exist in the Securities collection.

Show Details

Parameters		
PyObject	tickers	The symbols to retrieve historical data for.
DateTime	start	The start time in the algorithm's time zone.
DateTime	end	The end time in the algorithm's time zone.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - A python dictionary with a pandas DataFrame containing the requested historical data.

Definition at line 902 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 4/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
PyObject tickers,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution
)
```

Gets the historical data for the specified symbol between the specified dates. The symbol must exist in the Securities collection.

Show Details

Parameters		
PyObject	tickers	The symbols to retrieve historical data for.
DateTime	start	The start time in the algorithm's time zone.
DateTime	end	The end time in the algorithm's time zone.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.

Return

PyObject - A python dictionary with pandas DataFrame containing the requested historical data.

Definition at line 920 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 5/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
 PyObject type,
 PyObject tickers,
 DateTime start,
 DateTime end,
 *Nullable<Resolution> resolution,
 *Nullable<Boolean> fillForward,
 *Nullable<Boolean> extendedMarketHours,
 *Nullable<DataMappingMode> dataMappingMode,
 *Nullable<DataNormalizationMode> dataNormalizationMode,
 *Nullable<Int32> contractDepthOffset
 )
```

Gets the historical data for the specified symbols between the specified dates. The symbols must exist in the Securities collection.

Show Details

Parameters		
PyObject	type	The data type of the symbols.
PyObject	tickers	The symbols to retrieve historical data for.
DateTime	start	The start time in the algorithm's time zone.
DateTime	end	The end time in the algorithm's time zone.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - pandas.DataFrame containing the requested historical data.

Definition at line 942 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 6/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
 PyObject type,
 PyObject tickers,
 Int32 periods,
 *Nullable<Resolution> resolution,
 *Nullable<Boolean> fillForward,
 *Nullable<Boolean> extendedMarketHours,
 *Nullable<DataMappingMode> dataMappingMode,
 *Nullable<DataNormalizationMode> dataNormalizationMode,
 *Nullable<Int32> contractDepthOffset
 )
```

Gets the historical data for the specified symbols. The exact number of bars will be returned for each symbol. This may result in some data start earlier/later than others due to when various exchanges are open. The symbols must exist in the Securities collection.

Show Details

Parameters		
`PyObject`	type	The data type of the symbols.
`PyObject`	tickers	The symbols to retrieve historical data for.
`Int32`	periods	The number of bars to request.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`PyObject` - pandas.DataFrame containing the requested historical data.

Definition at line 970 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 7/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
 PyObject type,
 PyObject tickers,
 TimeSpan span,
 *Nullable<Resolution> resolution,
 *Nullable<Boolean> fillForward,
 *Nullable<Boolean> extendedMarketHours,
 *Nullable<DataMappingMode> dataMappingMode,
 *Nullable<DataNormalizationMode> dataNormalizationMode,
 *Nullable<Int32> contractDepthOffset
 )
```

Gets the historical data for the specified symbols over the requested span. The symbols must exist in the Securities collection.

Show Details

Parameters		
PyObject	type	The data type of the symbols.
PyObject	tickers	The symbols to retrieve historical data for.
TimeSpan	span	The span over which to retrieve recent historical data.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - pandas.DataFrame containing the requested historical data.

Definition at line 1000 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 8/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
PyObject type,
Symbol symbol,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbols between the specified dates. The symbols must exist in the Securities collection.

Show Details

Parameters		
`PyObject`	type	The data type of the symbols.
`Symbol`	symbol	The symbol to retrieve historical data for.
`DateTime`	start	The start time in the algorithm's time zone.
`DateTime`	end	The end time in the algorithm's time zone.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`PyObject` - pandas.DataFrame containing the requested historical data.

Definition at line 1024 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 9/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
PyObject type,
Symbol symbol,
Int32 periods,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbols. The exact number of bars will be returned for each symbol. This may result in some data start earlier/later than others due to when various exchanges are open. The symbols must exist in the Securities collection.

Show Details

Parameters		
PyObject	type	The data type of the symbols.
Symbol	symbol	The symbol to retrieve historical data for.
Int32	periods	The number of bars to request.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - pandas.DataFrame containing the requested historical data.

Definition at line 1057 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 10/27

```
PyObject QuantConnect.Algorithm.QCAlgorithm.History (
 PyObject type,
 Symbol symbol,
 TimeSpan span,
 *Nullable<Resolution> resolution,
 *Nullable<Boolean> fillForward,
 *Nullable<Boolean> extendedMarketHours,
 *Nullable<DataMappingMode> dataMappingMode,
 *Nullable<DataNormalizationMode> dataNormalizationMode,
 *Nullable<Int32> contractDepthOffset
 )
```

Gets the historical data for the specified symbols over the requested span. The symbols must exist in the

Securities collection.

Show Details

Parameters		
PyObject	type	The data type of the symbols.
Symbol	symbol	The symbol to retrieve historical data for.
TimeSpan	span	The span over which to retrieve recent historical data.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

PyObject - pandas.DataFrame containing the requested historical data.

Definition at line 1087 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA

History() 11/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
TimeSpan span,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Get the history for all configured securities over the requested span. This will use the resolution and other subscription settings for each security. The symbols must exist in the Securities collection.

Show Details

Parameters		
`TimeSpan`	span	The span over which to request data. This is a calendar span, so take into consideration weekends and such.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<Slice>` - An enumerable of slice containing data over the most recent span for all configured securities.

Definition at line 231 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 12/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
Int32 periods,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Get the history for all configured securities over the requested span. This will use the resolution and other subscription settings for each security. The symbols must exist in the Securities collection.

Show Details

Parameters		
`Int32`	periods	The number of bars to request.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<Slice>` - An enumerable of slice containing data over the most recent span for all configured securities.

Definition at line 253 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 13/27

```
IEnumerable<DataDictionary<T>> QuantConnect.Algorithm.QCAlgorithm.History (
TimeSpan span,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Get the history for all configured securities over the requested span. This will use the resolution and other subscription settings for each security. The symbols must exist in the Securities collection.

Show Details

Parameters		
TimeSpan	span	The span over which to request data. This is a calendar span, so take into consideration weekends and such.
*Nullable<Resolution>	resolution	*(Optional)* The resolution to request.
*Nullable<Boolean>	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
*Nullable<Boolean>	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
*Nullable<Int32>	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

IEnumerable<DataDictionary<T>> - An enumerable of slice containing data over the most recent span for all configured securities.

Definition at line 275 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 14/27

```
IEnumerable<DataDictionary<T>> QuantConnect.Algorithm.QCAlgorithm.History (
IEnumerable<Symbol> symbols,
TimeSpan span,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
TimeSpan	span	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Nullable<Boolean>	fillForward	*(Optional)* /
*Nullable<Boolean>	extendedMarketHours	*(Optional)* /
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* /
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* /
*Nullable<Int32>	contractDepthOffset	*(Optional)* /

Return

`IEnumerable<DataDictionary<T>>` - The new `IEnumerable<DataDictionary<T>>` object.

Definition at line 300 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 15/27

```
IEnumerable<DataDictionary<T>> QuantConnect.Algorithm.QCAlgorithm.History (
IEnumerable<Symbol> symbols,
Int32 periods,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`Int32`	periods	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Nullable<Boolean>`	fillForward	*(Optional)* /
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* /
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* /
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* /
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* /

Return

`IEnumerable<DataDictionary<T>>` - The new `IEnumerable<DataDictionary<T>>` object.

Definition at line 326 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 16/27

```
IEnumerable<DataDictionary<T>> QuantConnect.Algorithm.QCAlgorithm.History (
IEnumerable<Symbol> symbols,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`DateTime`	start	/
`DateTime`	end	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Nullable<Boolean>`	fillForward	*(Optional)* /
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* /
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* /
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* /
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* /

Return

`IEnumerable<DataDictionary<T>>` - The new `IEnumerable<DataDictionary<T>>` object.

Definition at line 353 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 17/27

```
IEnumerable<T> QuantConnect.Algorithm.QCAlgorithm.History (
Symbol symbol,
TimeSpan span,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
`Symbol`	symbol	The symbol to retrieve historical data for.
`TimeSpan`	span	The span over which to retrieve recent historical data.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<T>` - An enumerable of slice containing the requested historical data.

Definition at line 378 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 18/27

```
IEnumerable<TradeBar> QuantConnect.Algorithm.QCAlgorithm.History (
Symbol symbol,
Int32 periods,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol. The exact number of bars will be returned. The symbol must exist in the Securities collection.

Show Details

Parameters		
`Symbol`	symbol	The symbol to retrieve historical data for.
`Int32`	periods	The number of bars to request.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<TradeBar>` - An enumerable of slice containing the requested historical data.

Definition at line 402 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 19/27

```
IEnumerable<T> QuantConnect.Algorithm.QCAlgorithm.History (
Symbol symbol,
Int32 periods,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol. The exact number of bars will be returned. The symbol must exist in the Securities collection.

Show Details

Parameters		
`Symbol`	symbol	The symbol to retrieve historical data for.
`Int32`	periods	The number of bars to request.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<T>` - An enumerable of slice containing the requested historical data.

Definition at line 434 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 20/27

```
IEnumerable<T> QuantConnect.Algorithm.QCAlgorithm.History (
Symbol symbol,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
`Symbol`	symbol	The symbol to retrieve historical data for.
`DateTime`	start	The start time in the algorithm's time zone.
`DateTime`	end	The end time in the algorithm's time zone.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<T>` - An enumerable of slice containing the requested historical data.

Definition at line 461 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 21/27

```
IEnumerable<TradeBar> QuantConnect.Algorithm.QCAlgorithm.History (
Symbol symbol,
TimeSpan span,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
`Symbol`	symbol	The symbol to retrieve historical data for.
`TimeSpan`	span	The span over which to retrieve recent historical data.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<TradeBar>` - An enumerable of slice containing the requested historical data.

Definition at line 485 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 22/27

```
IEnumerable<TradeBar> QuantConnect.Algorithm.QCAlgorithm.History (
Symbol symbol,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Gets the historical data for the specified symbol over the request span. The symbol must exist in the Securities collection.

Show Details

Parameters		
`Symbol`	symbol	The symbol to retrieve historical data for.
`DateTime`	start	The start time in the algorithm's time zone.
`DateTime`	end	The end time in the algorithm's time zone.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution to request.
`*Nullable<Boolean>`	fillForward	*(Optional)* True to fill forward missing data, False otherwise.
`*Nullable<Boolean>`	extendedMarketHours	*(Optional)* True to include extended market hours data, False otherwise.
`*Nullable<DataMappingMode>`	dataMappingMode	*(Optional)* The contract mapping mode to use for the security history request.
`*Nullable<DataNormalizationMode>`	dataNormalizationMode	*(Optional)* The price scaling mode to use for the securities history.
`*Nullable<Int32>`	contractDepthOffset	*(Optional)* The continuous contract desired offset from the current front month. For example, 0 will use the front month, 1 will use the back month contract.

Return

`IEnumerable<TradeBar>` - An enumerable of slice containing the requested historical data.

Definition at line 508 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 23/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
IEnumerable<Symbol> symbols,
TimeSpan span,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Executes the specified history request.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
TimeSpan	span	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Nullable<Boolean>	fillForward	*(Optional)* /
*Nullable<Boolean>	extendedMarketHours	*(Optional)* /
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* /
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* /
*Nullable<Int32>	contractDepthOffset	*(Optional)* /

Return

IEnumerable<Slice> - The new IEnumerable<Slice> object.

Definition at line 545 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 24/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
IEnumerable<Symbol> symbols,
Int32 periods,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Executes the specified history request.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
Int32	periods	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Nullable<Boolean>	fillForward	*(Optional)* /
*Nullable<Boolean>	extendedMarketHours	*(Optional)* /
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* /
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* /
*Nullable<Int32>	contractDepthOffset	*(Optional)* /

Return

IEnumerable<Slice> - The new IEnumerable<Slice> object.

Definition at line 569 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 25/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
IEnumerable<Symbol> symbols,
DateTime start,
DateTime end,
*Nullable<Resolution> resolution,
*Nullable<Boolean> fillForward,
*Nullable<Boolean> extendedMarketHours,
*Nullable<DataMappingMode> dataMappingMode,
*Nullable<DataNormalizationMode> dataNormalizationMode,
*Nullable<Int32> contractDepthOffset
)
```

Executes the specified history request.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
DateTime	start	/
DateTime	end	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Nullable<Boolean>	fillForward	*(Optional)* /
*Nullable<Boolean>	extendedMarketHours	*(Optional)* /
*Nullable<DataMappingMode>	dataMappingMode	*(Optional)* /
*Nullable<DataNormalizationMode>	dataNormalizationMode	*(Optional)* /
*Nullable<Int32>	contractDepthOffset	*(Optional)* /

Return

IEnumerable<Slice> - The new IEnumerable<Slice> object.

Definition at line 593 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 26/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
 HistoryRequest request
)
```

Executes the specified history request.

Show Details

Parameters		
HistoryRequest	request	the history request to execute.

Return

IEnumerable<Slice> - An enumerable of slice satisfying the specified history request.

Definition at line 607 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

History() 27/27

```
IEnumerable<Slice> QuantConnect.Algorithm.QCAlgorithm.History (
 IEnumerable<HistoryRequest> requests
)
```

Show Details

Parameters		
IEnumerable<HistoryRequest>	requests	/

Return

IEnumerable<Slice> - The new IEnumerable<Slice> object.

Definition at line 618 of file Algorithm/QCAlgorithm.History.cs.

INDICATORS

HMA() 1/1

```
HullMovingAverage QuantConnect.Algorithm.QCAlgorithm.HMA (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new HullMovingAverage indicator. The Hull moving average is a series of nested weighted moving averages, is fast and smooth.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Hull moving average we want.
`Int32`	period	The period over which to compute the Hull moving average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`HullMovingAverage` - The new `HullMovingAverage` object.

Definition at line 712 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

HT() 1/1

```
HilbertTransform QuantConnect.Algorithm.QCAlgorithm.HT (
Symbol symbol,
Int32 length,
Decimal inPhaseMultiplicationFactor,
Decimal quadratureMultiplicationFactor,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Hilbert Transform indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Hilbert transform we want.
`Int32`	length	The length of the FIR filter used in the calculation of the Hilbert Transform. This parameter determines the number of filter coefficients in the FIR filter.
`Decimal`	inPhaseMultiplicationFactor	The multiplication factor used in the calculation of the in-phase component of the Hilbert Transform. This parameter adjusts the sensitivity and responsiveness of the transform to changes in the input signal.
`Decimal`	quadratureMultiplicationFactor	The multiplication factor used in the calculation of the quadrature component of the Hilbert Transform. This parameter also adjusts the sensitivity and responsiveness of the transform to changes in the input signal.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`HilbertTransform` - The new `HilbertTransform` object.

Definition at line 694 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ICHIMOKU() 1/1

```
IchimokuKinkoHyo QuantConnect.Algorithm.QCAlgorithm.ICHIMOKU (
Symbol symbol,
Int32 tenkanPeriod,
Int32 kijunPeriod,
Int32 senkouAPeriod,
Int32 senkouBPeriod,
Int32 senkouADelayPeriod,
Int32 senkouBDelayPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new IchimokuKinkoHyo indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose ICHIMOKU we want.
Int32	tenkanPeriod	The period to calculate the Tenkan-sen period.
Int32	kijunPeriod	The period to calculate the Kijun-sen period.
Int32	senkouAPeriod	The period to calculate the Tenkan-sen period.
Int32	senkouBPeriod	The period to calculate the Tenkan-sen period.
Int32	senkouADelayPeriod	The period to calculate the Tenkan-sen period.
Int32	senkouBDelayPeriod	The period to calculate the Tenkan-sen period.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, TradeBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

IchimokuKinkoHyo - A new IchimokuKinkoHyo indicator with the specified periods and delays.

Definition at line 736 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

Identity() 1/3

```
Identity QuantConnect.Algorithm.QCAlgorithm.Identity (
 Symbol symbol,
 *Func<IBaseData, Decimal> selector,
 *String fieldName
)
```

Creates a new Identity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`Identity` - A new Identity indicator for the specified symbol and selector.

Definition at line 755 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

Identity() 2/3

```
Identity QuantConnect.Algorithm.QCAlgorithm.Identity (
Symbol symbol,
Resolution resolution,
*Func<IBaseData, Decimal> selector,
*String fieldName
)
```

Creates a new Identity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose values we want as an indicator.
`Resolution`	resolution	The desired resolution of the data.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
`*String`	fieldName	*(Optional)* The name of the field being selected.

Return

`Identity` - A new Identity indicator for the specified symbol and selector.

Definition at line 771 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

Identity() 3/3

```
Identity QuantConnect.Algorithm.QCAlgorithm.Identity (
Symbol symbol,
TimeSpan resolution,
*Func<IBaseData, Decimal> selector,
*String fieldName
)
```

Creates a new Identity indicator for the symbol The indicator will be automatically updated on the symbol's subscription resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose values we want as an indicator.
TimeSpan	resolution	The desired resolution of the data.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData, if None defaults to the .Value property (x => x.Value).
*String	fieldName	*(Optional)* The name of the field being selected.

Return

Identity - A new Identity indicator for the specified symbol and selector.

Definition at line 789 of file Algorithm/QCAlgorithm.Indicators.cs.

ALGORITHM FRAMEWORK HANDLING DATA

Initialize() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.Initialize (

)
```

Initialise the data and resolution required, as well as the cash and start-end dates for your algorithm. All algorithms must initialized.

Show Details

This method requires no argument input.

Return

`Void` - This method provides no return.

Definition at line 584 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS | SECURITIES AND PORTFOLIO

IsMarketOpen() 1/1

```
Boolean QuantConnect.Algorithm.QCAlgorithm.IsMarketOpen (
 Symbol symbol
)
```

Determines if the exchange for the specified symbol is open at the current time.

Show Details

Parameters		
`Symbol`	symbol	The symbol.

Return

`Boolean` - True if the exchange is considered open at the current time, False otherwise.

Definition at line 1381 of file Algorithm/QCAlgorithm.Trading.cs.

HANDLING DATA | SECURITIES AND PORTFOLIO

ISIN() 1/2

```
Symbol QuantConnect.Algorithm.QCAlgorithm.ISIN (
String isin,
*Nullable<DateTime> tradingDate
)
```

Converts an ISIN identifier into a `Symbol` .

Show Details

Parameters		
`String`	isin	The International Securities Identification Number (ISIN) of an asset.
`*Nullable<DateTime>`	tradingDate	*(Optional)* The date that the stock being looked up is/was traded at. The date is used to create a Symbol with the ticker set to the ticker the asset traded under on the trading date. .

Return

`Symbol` - Symbol corresponding to the ISIN. If no Symbol with a matching ISIN was found, returns None.

Definition at line 2896 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA SECURITIES AND PORTFOLIO

ISIN() 2/2

```
String QuantConnect.Algorithm.QCAlgorithm.ISIN (
Symbol symbol
)
```

Converts a `Symbol` into an ISIN identifier.

Show Details

Parameters		
`Symbol`	symbol	The `Symbol` .

Return

`String` - ISIN corresponding to the Symbol. If no matching ISIN is found, returns None.

Definition at line 2908 of file Algorithm/QCAlgorithm.cs.

INDICATORS

KAMA() 1/2

```
KaufmanAdaptiveMovingAverage QuantConnect.Algorithm.QCAlgorithm.KAMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new KaufmanAdaptiveMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose KAMA we want.
`Int32`	period	The period of the Efficiency Ratio (ER) of KAMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`KaufmanAdaptiveMovingAverage` - The KaufmanAdaptiveMovingAverage indicator for the requested symbol over the specified period.

Definition at line 806 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

KAMA() 2/2

```
KaufmanAdaptiveMovingAverage QuantConnect.Algorithm.QCAlgorithm.KAMA (
 Symbol symbol,
 Int32 period,
 Int32 fastEmaPeriod,
 Int32 slowEmaPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new KaufmanAdaptiveMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose KAMA we want.
`Int32`	period	The period of the Efficiency Ratio (ER).
`Int32`	fastEmaPeriod	The period of the fast EMA used to calculate the Smoothing Constant (SC).
`Int32`	slowEmaPeriod	The period of the slow EMA used to calculate the Smoothing Constant (SC).
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`KaufmanAdaptiveMovingAverage` - The KaufmanAdaptiveMovingAverage indicator for the requested symbol over the specified period.

Definition at line 822 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

KCH() 1/1

```
KeltnerChannels QuantConnect.Algorithm.QCAlgorithm.KCH (
Symbol symbol,
Int32 period,
Decimal k,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Keltner Channels indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Keltner Channel we seek.
`Int32`	period	The period over which to compute the Keltner Channels.
`Decimal`	k	The number of multiples of the `AverageTrueRange` from the middle band of the Keltner Channels.
`*MovingAverageType`	movingAverageType	*(Optional)* Specifies the type of moving average to be used as the middle line of the Keltner Channel.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`KeltnerChannels` - The Keltner Channel indicator for the requested symbol.

Definition at line 862 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

KER() 1/1

```
KaufmanEfficiencyRatio QuantConnect.Algorithm.QCAlgorithm.KER (
Symbol symbol,
*Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an KaufmanEfficiencyRatio indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose EF we want.
`*Int32`	period	*(Optional)* The period of the EF.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`KaufmanEfficiencyRatio` - The KaufmanEfficiencyRatio indicator for the given parameters.

Definition at line 841 of file Algorithm/QCAlgorithm.Indicators.cs.

TRADING AND ORDERS

LimitIfTouchedOrder() 1/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.LimitIfTouchedOrder (
Symbol symbol,
Int32 quantity,
Decimal triggerPrice,
Decimal limitPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Send a limit if touched order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Int32`	quantity	Quantity of shares for limit order.
`Decimal`	triggerPrice	Trigger price for this order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 539 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

LimitIfTouchedOrder() 2/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.LimitIfTouchedOrder (
 Symbol symbol,
 Double quantity,
 Decimal triggerPrice,
 Decimal limitPrice,
 *String tag,
 *IOrderProperties orderProperties
 )
```

Send a limit if touched order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Double`	quantity	Quantity of shares for limit order.
`Decimal`	triggerPrice	Trigger price for this order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 555 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

LimitIfTouchedOrder() 3/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.LimitIfTouchedOrder (
 Symbol symbol,
 Decimal quantity,
 Decimal triggerPrice,
 Decimal limitPrice,
 *String tag,
 *IOrderProperties orderProperties
 )
```

Send a limit if touched order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Decimal`	quantity	Quantity of shares for limit order.
`Decimal`	triggerPrice	Trigger price for this order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 571 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

LimitOrder() 1/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.LimitOrder (
Symbol symbol,
Int32 quantity,
Decimal limitPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Send a limit order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Int32`	quantity	Quantity of shares for limit order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 391 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

LimitOrder() 2/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.LimitOrder (
Symbol symbol,
Double quantity,
Decimal limitPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Send a limit order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Double`	quantity	Quantity of shares for limit order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 406 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

LimitOrder() 3/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.LimitOrder (
Symbol symbol,
Decimal quantity,
Decimal limitPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Send a limit order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Decimal`	quantity	Quantity of shares for limit order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 421 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Liquidate() 1/1

```
List<Int32> QuantConnect.Algorithm.QCAlgorithm.Liquidate (
*Symbol symbolToLiquidate,
*String tag
)
```

Liquidate all holdings and cancel open orders. Called at the end of day for tick-strategies.

Show Details

Parameters		
`*Symbol`	symbolToLiquidate	*(Optional)* Symbols we wish to liquidate.
`*String`	tag	*(Optional)* Custom tag to know who is calling this.

Return

`List<Int32>` - Array of order Id for liquidated symbols.

Definition at line 1063 of file Algorithm/QCAlgorithm.Trading.cs.

LOGGING

Log() 1/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Log (
PyObject message
)
```

Added another method for logging if user guessed.

Show Details

Parameters		
PyObject	message	String message to log.

Return

Void - This method provides no return.

Definition at line 1234 of file Algorithm/QCAlgorithm.Python.cs.

LOGGING

Log() 2/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Log (
String message
)
```

Added another method for logging if user guessed.

Show Details

Parameters		
String	message	String message to log.

Return

`Void` - This method provides no return.

Definition at line 2503 of file Algorithm/QCAlgorithm.cs.

LOGGING

Log() 3/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Log (
 Int32 message
)
```

Added another method for logging if user guessed.

Show Details

Parameters		
`Int32`	message	Int message to log.

Return

`Void` - This method provides no return.

Definition at line 2516 of file Algorithm/QCAlgorithm.cs.

LOGGING

Log() 4/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Log (
 Double message
)
```

Added another method for logging if user guessed.

Show Details

Parameters		
Double	message	Double message to log.

Return

Void - This method provides no return.

Definition at line 2528 of file Algorithm/QCAlgorithm.cs.

LOGGING

Log() 5/5

```
Void QuantConnect.Algorithm.QCAlgorithm.Log (
Decimal message
)
```

Added another method for logging if user guessed.

Show Details

Parameters		
Decimal	message	Decimal message to log.

Return

Void - This method provides no return.

Definition at line 2540 of file Algorithm/QCAlgorithm.cs.

INDICATORS

LOGR() 1/1

```
LogReturn QuantConnect.Algorithm.QCAlgorithm.LOGR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new LogReturn indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose log return we seek.
`Int32`	period	The period of the log return.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`LogReturn` - log return indicator for the requested symbol.

Definition at line 880 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

LSMA() 1/1

```
LeastSquaresMovingAverage QuantConnect.Algorithm.QCAlgorithm.LSMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates and registers a new Least Squares Moving Average instance.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose LSMA we seek.
`Int32`	period	The LSMA period. Normally 14.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`LeastSquaresMovingAverage` - A LeastSquaredMovingAverage configured with the specified period.

Definition at line 898 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

LWMA() 1/1

```
LinearWeightedMovingAverage QuantConnect.Algorithm.QCAlgorithm.LWMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new LinearWeightedMovingAverage indicator. This indicator will linearly distribute the weights across the periods.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose LWMA we want.
`Int32`	period	The period over which to compute the LWMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`LinearWeightedMovingAverage` - The new `LinearWeightedMovingAverage` object.

Definition at line 917 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MACD() 1/1

```
MovingAverageConvergenceDivergence QuantConnect.Algorithm.QCAlgorithm.MACD (
Symbol symbol,
Int32 fastPeriod,
Int32 slowPeriod,
Int32 signalPeriod,
*MovingAverageType type,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a MACD indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose MACD we want.
Int32	fastPeriod	The period for the fast moving average.
Int32	slowPeriod	The period for the slow moving average.
Int32	signalPeriod	The period for the signal moving average.
*MovingAverageType	type	*(Optional)* The type of moving average to use for the MACD.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

MovingAverageConvergenceDivergence - The moving average convergence divergence between the fast and slow averages.

Definition at line 938 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MAD() 1/1

```
MeanAbsoluteDeviation QuantConnect.Algorithm.QCAlgorithm.MAD (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new MeanAbsoluteDeviation indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MeanAbsoluteDeviation we want.
`Int32`	period	The period over which to compute the MeanAbsoluteDeviation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MeanAbsoluteDeviation` - The MeanAbsoluteDeviation indicator for the requested symbol over the specified period.

Definition at line 956 of file Algorithm/QCAlgorithm.Indicators.cs.

TRADING AND ORDERS

MarketOnCloseOrder() 1/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOnCloseOrder (
 Symbol symbol,
 Int32 quantity,
 *String tag,
 *IOrderProperties orderProperties
)
```

Market on close order implementation: Send a market order when the exchange closes.

Show Details

Parameters		
`Symbol`	symbol	The symbol to be ordered.
`Int32`	quantity	The number of shares to required.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 342 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOnCloseOrder() 2/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOnCloseOrder (
Symbol symbol,
Double quantity,
*String tag,
*IOrderProperties orderProperties
)
```

Market on close order implementation: Send a market order when the exchange closes.

Show Details

Parameters		
`Symbol`	symbol	The symbol to be ordered.
`Double`	quantity	The number of shares to required.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 356 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOnCloseOrder() 3/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOnCloseOrder (
Symbol symbol,
Decimal quantity,
*String tag,
*IOrderProperties orderProperties
)
```

Market on close order implementation: Send a market order when the exchange closes.

Show Details

Parameters		
`Symbol`	symbol	The symbol to be ordered.
`Decimal`	quantity	The number of shares to required.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 370 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOnOpenOrder() 1/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOnOpenOrder (
Symbol symbol,
Double quantity,
*String tag,
*IOrderProperties orderProperties
)
```

Market on open order implementation: Send a market order when the exchange opens.

Show Details

Parameters		
`Symbol`	symbol	The symbol to be ordered.
`Double`	quantity	The number of shares to required.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 294 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOnOpenOrder() 2/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOnOpenOrder (
Symbol symbol,
Int32 quantity,
*String tag,
*IOrderProperties orderProperties
)
```

Market on open order implementation: Send a market order when the exchange opens.

Show Details

Parameters		
`Symbol`	symbol	The symbol to be ordered.
`Int32`	quantity	The number of shares to required.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 308 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOnOpenOrder() 3/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOnOpenOrder (
Symbol symbol,
Decimal quantity,
*String tag,
*IOrderProperties orderProperties
)
```

Market on open order implementation: Send a market order when the exchange opens.

Show Details

Parameters		
`Symbol`	symbol	The symbol to be ordered.
`Decimal`	quantity	The number of shares to required.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 322 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOrder() 1/4

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOrder (
Symbol symbol,
Int32 quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Market order implementation: Send a market order and wait for it to be filled.

Show Details

Parameters		
`Symbol`	symbol	Symbol of the MarketType Required.
`Int32`	quantity	Number of shares to request.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 206 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOrder() 2/4

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOrder (
Symbol symbol,
Double quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Market order implementation: Send a market order and wait for it to be filled.

Show Details

Parameters		
`Symbol`	symbol	Symbol of the MarketType Required.
`Double`	quantity	Number of shares to request.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 221 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOrder() 3/4

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOrder (
Symbol symbol,
Decimal quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Market order implementation: Send a market order and wait for it to be filled.

Show Details

Parameters		
`Symbol`	symbol	Symbol of the MarketType Required.
`Decimal`	quantity	Number of shares to request.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 236 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

MarketOrder() 4/4

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.MarketOrder (
Security security,
Decimal quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Market order implementation: Send a market order and wait for it to be filled.

Show Details

Parameters		
`Security`	security	Symbol of the MarketType Required.
`Decimal`	quantity	Number of shares to request.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 252 of file Algorithm/QCAlgorithm.Trading.cs.

INDICATORS

MASS() 1/1

```
MassIndex QuantConnect.Algorithm.QCAlgorithm.MASS (
 Symbol symbol,
 *Int32 emaPeriod,
 *Int32 sumPeriod,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Creates a new Mass Index indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Mass Index we want.
`*Int32`	emaPeriod	*(Optional)* The period used by both EMA.
`*Int32`	sumPeriod	*(Optional)* The sum period.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`MassIndex` - The Mass Index indicator for the requested symbol over the specified period.

Definition at line 1073 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MAX() 1/1

```
Maximum QuantConnect.Algorithm.QCAlgorithm.MAX (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new Maximum indicator to compute the maximum value.

Show Details

Parameters		
Symbol	symbol	The symbol whose max we want.
Int32	period	The look back period over which to compute the max value.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None and the symbol is of type TradeBar defaults to the High property, otherwise it defaults to Value property of BaseData (x => x.Value).

Return

Maximum - A Maximum indicator that compute the max value and the periods since the max value.

Definition at line 1017 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MFI() 1/1

```
MoneyFlowIndex QuantConnect.Algorithm.QCAlgorithm.MFI (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new MoneyFlowIndex indicator. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MFI we want.
`Int32`	period	The period over which to compute the MFI.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`MoneyFlowIndex` - The MoneyFlowIndex indicator for the requested symbol over the specified period.

Definition at line 1053 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MIDPOINT() 1/1

```
MidPoint QuantConnect.Algorithm.QCAlgorithm.MIDPOINT (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new MidPoint indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MIDPOINT we want.
`Int32`	period	The period over which to compute the MIDPOINT.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MidPoint` - The MidPoint indicator for the requested symbol over the specified period.

Definition at line 1091 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MIDPRICE() 1/1

```
MidPrice QuantConnect.Algorithm.QCAlgorithm.MIDPRICE (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new MidPrice indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose MIDPRICE we want.
`Int32`	period	The period over which to compute the MIDPRICE.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`MidPrice` - The MidPrice indicator for the requested symbol over the specified period.

Definition at line 1109 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MIN() 1/1

```
Minimum QuantConnect.Algorithm.QCAlgorithm.MIN (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Minimum indicator to compute the minimum value.

Show Details

Parameters		
Symbol	symbol	The symbol whose min we want.
Int32	period	The look back period over which to compute the min value.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None and the symbol is of type TradeBar defaults to the Low property, otherwise it defaults to Value property of BaseData (x => x.Value).

Return

Minimum - A Minimum indicator that compute the in value and the periods since the min value.

Definition at line 1128 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MOM() 1/1

```
Momentum QuantConnect.Algorithm.QCAlgorithm.MOM (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Momentum indicator. This will compute the absolute n-period change in the security. The indicator

will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose momentum we want.
`Int32`	period	The period over which to compute the momentum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Momentum` - The momentum indicator for the requested symbol over the specified period.

Definition at line 1164 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MOMERSION() 1/1

```
MomersionIndicator QuantConnect.Algorithm.QCAlgorithm.MOMERSION (
Symbol symbol,
Nullable<Int32> minPeriod,
Int32 fullPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Momersion indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Momersion we want.
`Nullable<Int32>`	minPeriod	The minimum period over which to compute the Momersion. Must be greater than 3. If None, only full period will be used in computations.
`Int32`	fullPeriod	The full period over which to compute the Momersion.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MomersionIndicator` - The Momersion indicator for the requested symbol over the specified period.

Definition at line 1183 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MOMP() 1/1

```
MomentumPercent QuantConnect.Algorithm.QCAlgorithm.MOMP (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new MomentumPercent indicator. This will compute the n-period percent change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose momentum we want.
`Int32`	period	The period over which to compute the momentum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`MomentumPercent` - The momentum indicator for the requested symbol over the specified period.

Definition at line 1202 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MOSC() 1/2

```
McClellanOscillator QuantConnect.Algorithm.QCAlgorithm.MOSC (
IEnumerable<Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Oscillator indicator.

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`*Int32`	fastPeriod	*(Optional)* /
`*Int32`	slowPeriod	*(Optional)* /
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* /

Return

`McClellanOscillator` - The new `McClellanOscillator` object.

Definition at line 2044 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MOSC() 2/2

```
McClellanOscillator QuantConnect.Algorithm.QCAlgorithm.MOSC (
Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Oscillator indicator.

Show Details

Parameters		
`Symbol>`	symbols	The symbols whose McClellan Oscillator we want.
`*Int32`	fastPeriod	*(Optional)* Fast period EMA of advance decline difference.
`*Int32`	slowPeriod	*(Optional)* Slow period EMA of advance decline difference.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`McClellanOscillator` - The McClellan Oscillator indicator for the requested symbol over the specified period.

Definition at line 2059 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MSI() 1/2

```
McClellanSummationIndex QuantConnect.Algorithm.QCAlgorithm.MSI (
IEnumerable<Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Summation Index indicator.

Show Details

Parameters		
IEnumerable<Symbol>	symbols	/
*Int32	fastPeriod	*(Optional)* /
*Int32	slowPeriod	*(Optional)* /
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, TradeBar>	selector	*(Optional)* /

Return

`McClellanSummationIndex` - The new `McClellanSummationIndex` object.

Definition at line 2082 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

MSI() 2/2

```
McClellanSummationIndex QuantConnect.Algorithm.QCAlgorithm.MSI (
Symbol> symbols,
*Int32 fastPeriod,
*Int32 slowPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new McClellan Summation Index indicator.

Show Details

Parameters		
Symbol>	symbols	The symbols whose McClellan Summation Index we want.
*Int32	fastPeriod	*(Optional)* Fast period EMA of advance decline difference.
*Int32	slowPeriod	*(Optional)* Slow period EMA of advance decline difference.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, TradeBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`McClellanSummationIndex` - The McClellan Summation Index indicator for the requested symbol over the specified period.

Definition at line 2097 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

NATR() 1/1

```
NormalizedAverageTrueRange QuantConnect.Algorithm.QCAlgorithm.NATR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new NormalizedAverageTrueRange indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose NATR we want.
Int32	period	The period over which to compute the NATR.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, IBaseDataBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

NormalizedAverageTrueRange - The NormalizedAverageTrueRange indicator for the requested symbol over the specified period.

Definition at line 1220 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

OBV() 1/1

```
OnBalanceVolume QuantConnect.Algorithm.QCAlgorithm.OBV (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new On Balance Volume indicator. This will compute the cumulative total volume based on whether the close price being higher or lower than the previous period. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose On Balance Volume we seek.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`OnBalanceVolume` - The On Balance Volume indicator for the requested symbol.

Definition at line 1239 of file Algorithm/QCAlgorithm.Indicators.cs.

TRADING AND ORDERS

OnAssignmentOrderEvent() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnAssignmentOrderEvent (
OrderEvent assignmentEvent
)
```

Option assignment event handler. On an option assignment event for short legs the resulting information is passed to this method.

Show Details

Parameters		
`OrderEvent`	assignmentEvent	Option exercise event details containing details of the assignment.

Return

`Void` - This method provides no return.

Definition at line 1045 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnData() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnData (
 Slice slice
)
```

Event - v3.0 DATA EVENT HANDLER: (Pattern) Basic template for user to override for receiving all subscription data in a single event.

Show Details

Parameters		
Slice	slice	The current slice of data keyed by symbol string.

Return

Void - This method provides no return.

Definition at line 857 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnEndOfAlgorithm() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnEndOfAlgorithm (
)
```

End of algorithm run event handler. This method is called at the end of a backtest or live trading operation. Intended for closing out logs.

Show Details

This method requires no argument input.

Return

Void - This method provides no return.

Definition at line 1023 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnEndOfDay() 1/3

```
Void QuantConnect.Algorithm.QCAlgorithm.OnEndOfDay (
)
```

End of a trading day event handler. This method is called at the end of the algorithm day (or multiple times if trading multiple assets).

Show Details

This method requires no argument input.

Return

`Void` - This method provides no return.

Definition at line 991 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnEndOfDay() 2/3

```
Void QuantConnect.Algorithm.QCAlgorithm.OnEndOfDay (
 String symbol
)
```

End of a trading day event handler. This method is called at the end of the algorithm day (or multiple times if trading multiple assets).

Show Details

Parameters		
`String`	symbol	Asset symbol for this end of day event. Forex and equities have different closing hours.

Return

`Void` - This method provides no return.

Definition at line 1005 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnEndOfDay() 3/3

```
Void QuantConnect.Algorithm.QCAlgorithm.OnEndOfDay (
 Symbol symbol
)
```

End of a trading day event handler. This method is called at the end of the algorithm day (or multiple times if trading multiple assets).

Show Details

Parameters		
`Symbol`	symbol	Asset symbol for this end of day event. Forex and equities have different closing hours.

Return

`Void` - This method provides no return.

Definition at line 1014 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnEndOfTimeStep() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnEndOfTimeStep (

)
```

Invoked at the end of every time step. This allows the algorithm to process events before advancing to the next time step.

Show Details

This method requires no argument input.

Return

`Void` - This method provides no return.

Definition at line 68 of file Algorithm/QCAlgorithm.Universe.cs.

ALGORITHM FRAMEWORK HANDLING DATA

OnFrameworkData() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnFrameworkData (
Slice slice
)
```

Used to send data updates to algorithm framework models.

Show Details

Parameters		
`Slice`	slice	The current data slice.

Return

`Void` - This method provides no return.

Definition at line 107 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK UNIVERSES

OnFrameworkSecuritiesChanged() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnFrameworkSecuritiesChanged (
SecurityChanges changes
)
```

Used to send security changes to algorithm framework models.

Show Details

Parameters		
SecurityChanges	changes	Security additions/removals for this time step.

Return

Void - This method provides no return.

Definition at line 260 of file Algorithm/QCAlgorithm.Framework.cs.

MODELING TRADING AND ORDERS

OnMarginCall() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnMarginCall (
List<SubmitOrderRequest> requests
)
```

Margin call event handler. This method is called right before the margin call orders are placed in the market.

Show Details

Parameters		
List<SubmitOrderRequest>	requests	The orders to be executed to bring this algorithm within margin limits.

Return

Void - This method provides no return.

Definition at line 969 of file Algorithm/QCAlgorithm.cs.

MODELING TRADING AND ORDERS

OnMarginCallWarning() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnMarginCallWarning (
)
```

Margin call warning event handler. This method is called when Portfolio.MarginRemaining is under 5% of your Portfolio.TotalPortfolioValue.

Show Details

This method requires no argument input.

Return

`Void` - This method provides no return.

Definition at line 978 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

OnOrderEvent() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnOrderEvent (
 OrderEvent orderEvent
)
```

Order fill event handler. On an order fill update the resulting information is passed to this method.

Show Details

Parameters		
`OrderEvent`	orderEvent	Order event details containing details of the events.

Return

`Void` - This method provides no return.

Definition at line 1034 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

OnWarmupFinished() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.OnWarmupFinished (
)
```

Called when the algorithm has completed initialization and warm up.

Show Details

This method requires no argument input.

Return

`Void` - This method provides no return.

Definition at line 668 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

Order() 1/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
 Symbol symbol,
 Double quantity
)
```

Issue an order/trade for asset: Alias wrapper for Order(string, int);.

Show Details

Parameters		
`Symbol`	symbol	Symbol to order.
`Double`	quantity	Quantity to order.

Return

`OrderTicket` - The order ticket instance.

Definition at line 151 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 2/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
Symbol symbol,
Int32 quantity
)
```

Issue an order/trade for asset.

Show Details

Parameters		
Symbol	symbol	Symbol to order.
Int32	quantity	Quantity to order.

Return

OrderTicket - The order ticket instance.

Definition at line 163 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 3/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
Symbol symbol,
Decimal quantity
)
```

Issue an order/trade for asset.

Show Details

Parameters		
`Symbol`	symbol	Symbol to order.
`Decimal`	quantity	Quantity to order.

Return

`OrderTicket` - The order ticket instance.

Definition at line 175 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 4/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
Symbol symbol,
Decimal quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Wrapper for market order method: submit a new order for quantity of symbol using type order.

Show Details

Parameters		
`Symbol`	symbol	Symbol of the MarketType Required.
`Decimal`	quantity	Number of shares to request.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* Place a custom order property or tag (e.g. indicator data).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 191 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 5/8

```
IEnumerable<OrderTicket> QuantConnect.Algorithm.QCAlgorithm.Order (
OptionStrategy strategy,
Int32 quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Issue an order/trade for buying/selling an option strategy.

Show Details

Parameters		
`OptionStrategy`	strategy	Specification of the strategy to trade.
`Int32`	quantity	Quantity of the strategy to trade.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`IEnumerable<OrderTicket>` - Sequence of order tickets.

Definition at line 658 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 6/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
Symbol symbol,
Int32 quantity,
OrderType type,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Obsolete implementation of Order method accepting a OrderType. This was deprecated since it was impossible to generate other orders via this method. Any calls to this method will always default to a Market Order.

Show Details

Parameters		
`Symbol`	symbol	Symbol we want to purchase.
`Int32`	quantity	Quantity to buy, + is long, - short.
`OrderType`	type	Order Type. *Options: ['Market', 'Limit', 'StopMarket', 'StopLimit', 'MarketOnOpen', 'MarketOnClose', 'OptionExercise', 'LimitIfTouched', 'ComboMarket', 'ComboLimit', 'ComboLegLimit']*
`*Boolean`	asynchronous	*(Optional)* Don't wait for the response, just submit order and move on.
`*String`	tag	*(Optional)* Custom data for this order.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 1342 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 7/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
Symbol symbol,
Decimal quantity,
OrderType type
)
```

Obsolete method for placing orders.

Show Details

Parameters		
`Symbol`	symbol	Symbol we want to order.
`Decimal`	quantity	The quantity to order.
`OrderType`	type	The order type.

Return

`OrderTicket` - The order ticket instance.

Definition at line 1356 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Order() 8/8

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Order (
Symbol symbol,
Int32 quantity,
OrderType type
)
```

Obsolete method for placing orders.

Show Details

Parameters		
`Symbol`	symbol	Symbol we want to order.
`Int32`	quantity	The quantity to order.
`OrderType`	type	The order type.

Return

`OrderTicket` - The order ticket instance.

Definition at line 1370 of file Algorithm/QCAlgorithm.Trading.cs.

CHARTING

Plot() 1/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String chart,
String series,
Int32 value
)
```

Plot a chart to string chart name, using string series name, with int value.

Show Details

Parameters		
String	chart	/
String	series	/
Int32	value	/

Return

Void - This method provides no return.

Definition at line 149 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 2/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String chart,
String series,
Single value
)
```

Plot a chart to string chart name, using string series name, with float value.

Show Details

Parameters		
String	chart	/
String	series	/
Single	value	/

Return

Void - This method provides no return.

Definition at line 159 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 3/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
 String chart,
 String series,
 Decimal value
)
```

Plot a value to a chart of string-chart name, with string series name, and decimal value. If chart does not exist, create it.

Show Details

Parameters		
String	chart	Chart name.
String	series	Series name.
Decimal	value	Value of the point.

Return

Void - This method provides no return.

Definition at line 171 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 4/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String chart,
IndicatorBase> indicators
)
```

Plots the value of each indicator on the chart.

Show Details

Parameters		
String	chart	The chart's name.
IndicatorBase>	indicators	The indicators to plot.

Return

Void - This method provides no return.

Definition at line 249 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 5/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String series,
PyObject pyObject
)
```

Plot a chart using string series name, with value.

Show Details

Parameters		
String	series	Name of the plot series.
PyObject	pyObject	PyObject with the value to plot.

Return

Void - This method provides no return.

Definition at line 690 of file Algorithm/QCAlgorithm.Python.cs.

CHARTING

Plot() 6/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
 String chart,
 Indicator first,
 *Indicator second,
 *Indicator third,
 *Indicator fourth
 )
```

Plots the value of each indicator on the chart.

Show Details

Parameters		
String	chart	The chart's name.
Indicator	first	The first indicator to plot.
*Indicator	second	*(Optional)* The second indicator to plot.
*Indicator	third	*(Optional)* The third indicator to plot.
*Indicator	fourth	*(Optional)* The fourth indicator to plot.

Return

Void - This method provides no return.

Definition at line 724 of file Algorithm/QCAlgorithm.Python.cs.

CHARTING

Plot() 7/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String chart,
BarIndicator first,
*BarIndicator second,
*BarIndicator third,
*BarIndicator fourth
)
```

Plots the value of each indicator on the chart.

Show Details

Parameters		
String	chart	The chart's name.
BarIndicator	first	The first indicator to plot.
*BarIndicator	second	*(Optional)* The second indicator to plot.
*BarIndicator	third	*(Optional)* The third indicator to plot.
*BarIndicator	fourth	*(Optional)* The fourth indicator to plot.

Return

Void - This method provides no return.

Definition at line 739 of file Algorithm/QCAlgorithm.Python.cs.

CHARTING

Plot() 8/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String chart,
TradeBarIndicator first,
*TradeBarIndicator second,
*TradeBarIndicator third,
*TradeBarIndicator fourth
)
```

Plots the value of each indicator on the chart.

Show Details

Parameters		
String	chart	The chart's name.
TradeBarIndicator	first	The first indicator to plot.
*TradeBarIndicator	second	*(Optional)* The second indicator to plot.
*TradeBarIndicator	third	*(Optional)* The third indicator to plot.
*TradeBarIndicator	fourth	*(Optional)* The fourth indicator to plot.

Return

Void - This method provides no return.

Definition at line 754 of file Algorithm/QCAlgorithm.Python.cs.

CHARTING

Plot() 9/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
 String series,
 Decimal value
)
```

Plot a chart using string series name, with value.

Show Details

Parameters		
String	series	Name of the plot series.
Decimal	value	Value to plot.

Return

Void - This method provides no return.

Definition at line 65 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 10/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String series,
Double value
)
```

Plot a chart using string series name, with double value.

Show Details

Parameters		
String	series	/
Double	value	/

Return

Void - This method provides no return.

Definition at line 110 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 11/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String series,
Int32 value
)
```

Plot a chart using string series name, with int value.

Show Details

Parameters		
String	series	/
Int32	value	/

Return

Void - This method provides no return.

Definition at line 119 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 12/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String series,
Single value
)
```

Plot a chart using string series name, with float value.

Show Details

Parameters		
String	series	/
Single	value	/

Return

Void - This method provides no return.

Definition at line 129 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Plot() 13/13

```
Void QuantConnect.Algorithm.QCAlgorithm.Plot (
String chart,
String series,
Double value
)
```

Plot a chart to string chart name, using string series name, with double value.

Show Details

Parameters		
String	chart	/
String	series	/
Double	value	/

Return

Void - This method provides no return.

Definition at line 139 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING INDICATORS

PlotIndicator() 1/4

```
Void QuantConnect.Algorithm.QCAlgorithm.PlotIndicator (
String chart,
IndicatorBase> indicators
)
```

Automatically plots each indicator when a new value is available.

Show Details

Parameters		
String	chart	/
IndicatorBase>	indicators	/

Return

Void - This method provides no return.

Definition at line 261 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING INDICATORS

PlotIndicator() 2/4

```
Void QuantConnect.Algorithm.QCAlgorithm.PlotIndicator (
String chart,
Boolean waitForReady,
IndicatorBase> indicators
)
```

Automatically plots each indicator when a new value is available, optionally waiting for indicator.IsReady to return True.

Show Details

Parameters		
String	chart	/
Boolean	waitForReady	/
IndicatorBase>	indicators	/

Return

Void - This method provides no return.

Definition at line 271 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING INDICATORS

PlotIndicator() 3/4

```
Void QuantConnect.Algorithm.QCAlgorithm.PlotIndicator (
String chart,
PyObject first,
*PyObject second,
*PyObject third,
*PyObject fourth
)
```

Automatically plots each indicator when a new value is available.

Show Details

Parameters		
String	chart	/
PyObject	first	/
*PyObject	second	*(Optional)* /
*PyObject	third	*(Optional)* /
*PyObject	fourth	*(Optional)* /

Return

Void - This method provides no return.

Definition at line 763 of file Algorithm/QCAlgorithm.Python.cs.

CHARTING INDICATORS

PlotIndicator() 4/4

```
Void QuantConnect.Algorithm.QCAlgorithm.PlotIndicator (
String chart,
Boolean waitForReady,
PyObject first,
*PyObject second,
*PyObject third,
*PyObject fourth
)
```

Automatically plots each indicator when a new value is available.

Show Details

Parameters		
`String`	chart	/
`Boolean`	waitForReady	/
`PyObject`	first	/
`*PyObject`	second	*(Optional)* /
`*PyObject`	third	*(Optional)* /
`*PyObject`	fourth	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 774 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS

PPHL() 1/1

```
PivotPointsHighLow QuantConnect.Algorithm.QCAlgorithm.PPHL (
 Symbol symbol,
 Int32 lengthHigh,
 Int32 lengthLow,
 *Int32 lastStoredValues,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new PivotPointsHighLow indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose PPHL we seek.
Int32	lengthHigh	The number of surrounding bars whose high values should be less than the current bar's for the bar high to be marked as high pivot point.
Int32	lengthLow	The number of surrounding bars whose low values should be more than the current bar's for the bar low to be marked as low pivot point.
*Int32	lastStoredValues	*(Optional)* The number of last stored indicator values.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, IBaseDataBar>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

PivotPointsHighLow - The PivotPointsHighLow indicator for the requested symbol.

Definition at line 1259 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

PPO() 1/1

```
PercentagePriceOscillator QuantConnect.Algorithm.QCAlgorithm.PPO (
 Symbol symbol,
 Int32 fastPeriod,
 Int32 slowPeriod,
 MovingAverageType movingAverageType,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new PercentagePriceOscillator indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose PPO we want.
Int32	fastPeriod	The fast moving average period.
Int32	slowPeriod	The slow moving average period.
MovingAverageType	movingAverageType	The type of moving average to use.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

PercentagePriceOscillator - The PercentagePriceOscillator indicator for the requested symbol over the specified period.

Definition at line 1279 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

PSAR() 1/1

```
ParabolicStopAndReverse QuantConnect.Algorithm.QCAlgorithm.PSAR (
 Symbol symbol,
 *Decimal afStart,
 *Decimal afIncrement,
 *Decimal afMax,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, IBaseDataBar> selector
 )
```

Creates a new Parabolic SAR indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose PSAR we seek.
`*Decimal`	afStart	*(Optional)* Acceleration factor start value. Normally 0.02.
`*Decimal`	afIncrement	*(Optional)* Acceleration factor increment value. Normally 0.02.
`*Decimal`	afMax	*(Optional)* Acceleration factor max value. Normally 0.2.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`ParabolicStopAndReverse` - A ParabolicStopAndReverse configured with the specified periods.

Definition at line 1299 of file Algorithm/QCAlgorithm.Indicators.cs.

LOGGING

Quit() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.Quit (
 PyObject message
)
```

Terminate the algorithm after processing the current event handler.

Show Details

Parameters		
`PyObject`	message	Exit message to display on quitting.

Return

`Void` - This method provides no return.

Definition at line 1244 of file Algorithm/QCAlgorithm.Python.cs.

LOGGING

Quit() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.Quit (
*String message
)
```

Terminate the algorithm after processing the current event handler.

Show Details

Parameters		
*String	message	*(Optional)* Exit message to display on quitting.

Return

Void - This method provides no return.

Definition at line 2615 of file Algorithm/QCAlgorithm.cs.

INDICATORS

RC() 1/1

```
RegressionChannel QuantConnect.Algorithm.QCAlgorithm.RC (
Symbol symbol,
Int32 period,
Decimal k,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RegressionChannel indicator which will compute the LinearRegression, UpperChannel and LowerChannel lines, the intercept and slope.

Show Details

Parameters		
Symbol	symbol	The symbol whose RegressionChannel we seek.
Int32	period	The period of the standard deviation and least square moving average (linear regression line).
Decimal	k	The number of standard deviations specifying the distance between the linear regression and upper or lower channel lines.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RegressionChannel` - A Regression Channel configured with the specified period and number of standard deviation.

Definition at line 1318 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

RDV() 1/1

```
RelativeDailyVolume QuantConnect.Algorithm.QCAlgorithm.RDV (
Symbol symbol,
*Int32 period,
*Resolution resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an RelativeDailyVolume indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RDV we want.
`*Int32`	period	*(Optional)* The period of the RDV.
`*Resolution`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`RelativeDailyVolume` - The Relative Volume indicator for the given parameters.

Definition at line 1450 of file Algorithm/QCAlgorithm.Indicators.cs.

CHARTING

Record() 1/3

```
Void QuantConnect.Algorithm.QCAlgorithm.Record (
 String series,
 Int32 value
)
```

Plot a chart using string series name, with int value. Alias of Plot();.

Show Details

Parameters		
`String`	series	/
`Int32`	value	/

Return

`Void` - This method provides no return.

Definition at line 77 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Record() 2/3

```
Void QuantConnect.Algorithm.QCAlgorithm.Record (
String series,
Double value
)
```

Plot a chart using string series name, with double value. Alias of Plot();.

Show Details

Parameters		
String	series	/
Double	value	/

Return

Void - This method provides no return.

Definition at line 87 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

Record() 3/3

```
Void QuantConnect.Algorithm.QCAlgorithm.Record (
String series,
Decimal value
)
```

Plot a chart using string series name, with decimal value. Alias of Plot();.

Show Details

Parameters		
String	series	/
Decimal	value	/

Return

Void - This method provides no return.

Definition at line 99 of file Algorithm/QCAlgorithm.Plotting.cs.

INDICATORS CONSOLIDATING DATA

RegisterIndicator() 1/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
PyObject indicator,
*Nullable<Resolution> resolution,
*PyObject selector
)
```

Registers the consolidator to receive automatic updates as well as configures the indicator to receive updates from the consolidator.

Show Details

Parameters		
Symbol	symbol	The symbol to register against.
PyObject	indicator	The indicator to receive data from the consolidator.
*Nullable<Resolution>	resolution	*(Optional)* The resolution at which to send data to the indicator, None to use the same resolution as the subscription.
*PyObject	selector	*(Optional)* Selects a value from the BaseData send into the indicator, if None defaults to a cast (x => (T)x).

Return

Void - This method provides no return.

Definition at line 545 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS CONSOLIDATING DATA

RegisterIndicator() 2/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
PyObject indicator,
*Nullable<TimeSpan> resolution,
*PyObject selector
)
```

Registers the consolidator to receive automatic updates as well as configures the indicator to receive updates from the consolidator.

Show Details

Parameters		
Symbol	symbol	The symbol to register against.
PyObject	indicator	The indicator to receive data from the consolidator.
*Nullable<TimeSpan>	resolution	*(Optional)* The resolution at which to send data to the indicator, None to use the same resolution as the subscription.
*PyObject	selector	*(Optional)* Selects a value from the BaseData send into the indicator, if None defaults to a cast (x => (T)x).

Return

Void - This method provides no return.

Definition at line 560 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS CONSOLIDATING DATA

RegisterIndicator() 3/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
PyObject indicator,
PyObject pyObject,
*PyObject selector
)
```

Registers the consolidator to receive automatic updates as well as configures the indicator to receive updates from the consolidator.

Show Details

Parameters		
Symbol	symbol	The symbol to register against.
PyObject	indicator	The indicator to receive data from the consolidator.
PyObject	pyObject	The python object that it is trying to register with, could be consolidator or a timespan.
*PyObject	selector	*(Optional)* Selects a value from the BaseData send into the indicator, if None defaults to a cast (x => (T)x).

Return

Void - This method provides no return.

Definition at line 575 of file Algorithm/QCAlgorithm.Python.cs.

INDICATORS CONSOLIDATING DATA

RegisterIndicator() 4/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
PyObject indicator,
IDataConsolidator consolidator,
*PyObject selector
)
```

Registers the consolidator to receive automatic updates as well as configures the indicator to receive updates from the consolidator.

Show Details

Parameters		
Symbol	symbol	The symbol to register against.
PyObject	indicator	The indicator to receive data from the consolidator.
IDataConsolidator	consolidator	The consolidator to receive raw subscription data.
*PyObject	selector	*(Optional)* Selects a value from the BaseData send into the indicator, if None defaults to a cast (x => (T)x).

Return

Void - This method provides no return.

Definition at line 624 of file Algorithm/QCAlgorithm.Python.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 5/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
IndicatorBase<IndicatorDataPoint> indicator,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Show Details

Parameters		
Symbol	symbol	/
IndicatorBase<IndicatorDataPoint>	indicator	/
*Nullable<Resolution>	resolution	*(Optional)* /
*Func<IBaseData, Decimal>	selector	*(Optional)* /

Return

Void - This method provides no return.

Definition at line 2224 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 6/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
IndicatorBase<IndicatorDataPoint> indicator,
*Nullable<TimeSpan> resolution,
*Func<IBaseData, Decimal> selector
)
```

Show Details

Parameters		
Symbol	symbol	/
IndicatorBase<IndicatorDataPoint>	indicator	/
*Nullable<TimeSpan>	resolution	*(Optional)* /
*Func<IBaseData, Decimal>	selector	*(Optional)* /

Return

Void - This method provides no return.

Definition at line 2239 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 7/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
IndicatorBase<IndicatorDataPoint> indicator,
IDataConsolidator consolidator,
*Func<IBaseData, Decimal> selector
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<IndicatorDataPoint>`	indicator	/
`IDataConsolidator`	consolidator	/
`*Func<IBaseData, Decimal>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2254 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 8/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
IndicatorBase<T> indicator,
*Nullable<Resolution> resolution
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<T>`	indicator	/
`*Nullable<Resolution>`	resolution	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2279 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 9/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
IndicatorBase<T> indicator,
Nullable<Resolution> resolution,
Func<IBaseData, T> selector
)
```

Show Details

Parameters		
Symbol	symbol	/
IndicatorBase<T>	indicator	/
Nullable<Resolution>	resolution	/
Func<IBaseData, T>	selector	/

Return

Void - This method provides no return.

Definition at line 2295 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 10/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
Symbol symbol,
IndicatorBase<T> indicator,
Nullable<TimeSpan> resolution,
*Func<IBaseData, T> selector
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<T>`	indicator	/
`Nullable<TimeSpan>`	resolution	/
`*Func<IBaseData, T>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2311 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA INDICATORS

RegisterIndicator() 11/11

```
Void QuantConnect.Algorithm.QCAlgorithm.RegisterIndicator (
 Symbol symbol,
 IndicatorBase<T> indicator,
 IDataConsolidator consolidator,
 *Func<IBaseData, T> selector
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<T>`	indicator	/
`IDataConsolidator`	consolidator	/
`*Func<IBaseData, T>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2327 of file Algorithm/QCAlgorithm.Indicators.cs.

ADDING DATA

RemoveOptionContract() 1/1

```
Boolean QuantConnect.Algorithm.QCAlgorithm.RemoveOptionContract (
 Symbol symbol
)
```

Removes the security with the specified symbol. This will cancel all open orders and then liquidate any existing holdings.

Show Details

Parameters		
Symbol	symbol	The symbol of the security to be removed.

Return

Boolean - The new Boolean object.

Definition at line 2225 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

RemoveSecurity() 1/1

```
Boolean QuantConnect.Algorithm.QCAlgorithm.RemoveSecurity (
 Symbol symbol
)
```

Removes the security with the specified symbol. This will cancel all open orders and then liquidate any existing holdings.

Show Details

Parameters		
Symbol	symbol	The symbol of the security to be removed.

Return

`Boolean` - The new `Boolean` object.

Definition at line 2236 of file Algorithm/QCAlgorithm.cs.

CONSOLIDATING DATA | INDICATORS

ResolveConsolidator() 1/2

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.ResolveConsolidator (
Symbol symbol,
Nullable<Resolution> resolution,
*Type dataType
)
```

Gets the default consolidator for the specified symbol and resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose data is to be consolidated.
`Nullable<Resolution>`	resolution	The resolution for the consolidator, if None, uses the resolution from subscription.
`*Type`	dataType	*(Optional)* The data type for this consolidator, if None, uses TradeBar over QuoteBar if present.

Return

`IDataConsolidator` - The new default consolidator.

Definition at line 2550 of file Algorithm/QCAlgorithm.Indicators.cs.

CONSOLIDATING DATA | INDICATORS

ResolveConsolidator() 2/2

```
IDataConsolidator QuantConnect.Algorithm.QCAlgorithm.ResolveConsolidator (
Symbol symbol,
Nullable<TimeSpan> timeSpan,
*Type dataType
)
```

Gets the default consolidator for the specified symbol and resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose data is to be consolidated.
`Nullable<TimeSpan>`	timeSpan	The requested time span for the consolidator, if None, uses the resolution from subscription.
`*Type`	dataType	*(Optional)* The data type for this consolidator, if None, uses TradeBar over QuoteBar if present.

Return

`IDataConsolidator` - The new default consolidator.

Definition at line 2569 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

RMA() 1/1

```
RelativeMovingAverage QuantConnect.Algorithm.QCAlgorithm.RMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Relative Moving Average indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose relative moving average we seek.
`Int32`	period	The period of the relative moving average.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RelativeMovingAverage` - A relative moving average configured with the specified period and number of standard deviation.

Definition at line 1336 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ROC() 1/1

```
RateOfChange QuantConnect.Algorithm.QCAlgorithm.ROC (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RateOfChange indicator. This will compute the n-period rate of change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose RateOfChange we want.
Int32	period	The period over which to compute the RateOfChange.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RateOfChange` - The RateOfChange indicator for the requested symbol over the specified period.

Definition at line 1356 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ROCP() 1/1

```
RateOfChangePercent QuantConnect.Algorithm.QCAlgorithm.ROCP (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RateOfChangePercent indicator. This will compute the n-period percentage rate of change in the security. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose RateOfChangePercent we want.
Int32	period	The period over which to compute the RateOfChangePercent.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

RateOfChangePercent - The RateOfChangePercent indicator for the requested symbol over the specified period.

Definition at line 1375 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ROCR() 1/1

```
RateOfChangeRatio QuantConnect.Algorithm.QCAlgorithm.ROCR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RateOfChangeRatio indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose ROCR we want.
Int32	period	The period over which to compute the ROCR.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RateOfChangeRatio` - The RateOfChangeRatio indicator for the requested symbol over the specified period.

Definition at line 1393 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

RSI() 1/1

```
RelativeStrengthIndex QuantConnect.Algorithm.QCAlgorithm.RSI (
Symbol symbol,
Int32 period,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RelativeStrengthIndex indicator. This will produce an oscillator that ranges from 0 to 100 based on the ratio of average gains to average losses over the specified period.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RSI we want.
`Int32`	period	The period over which to compute the RSI.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use in computing the average gain/loss values.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`RelativeStrengthIndex` - The RelativeStrengthIndex indicator for the requested symbol over the specified period.

Definition at line 1413 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

RVI() 1/1

```
RelativeVigorIndex QuantConnect.Algorithm.QCAlgorithm.RVI (
Symbol symbol,
Int32 period,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new RelativeVigorIndex indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose RVI we want.
`Int32`	period	The period over which to compute the RVI.
`*MovingAverageType`	movingAverageType	*(Optional)* The type of moving average to use.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`RelativeVigorIndex` - The RelativeVigorIndex indicator for the requested symbol over the specified period.

Definition at line 1431 of file Algorithm/QCAlgorithm.Indicators.cs.

HANDLING DATA SECURITIES AND PORTFOLIO

SEDOL() 1/2

```
Symbol QuantConnect.Algorithm.QCAlgorithm.SEDOL (
String sedol,
*Nullable<DateTime> tradingDate
)
```

Converts a SEDOL identifier into a `Symbol` .

Show Details

Parameters		
`String`	sedol	The SEDOL identifier of an asset.
`*Nullable< DateTime>`	tradingDate	*(Optional)* The date that the stock being looked up is/was traded at. The date is used to create a Symbol with the ticker set to the ticker the asset traded under on the trading date. .

Return

`Symbol` - Symbol corresponding to the SEDOL. If no Symbol with a matching SEDOL was found, returns None.

Definition at line 2984 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA SECURITIES AND PORTFOLIO

SEDOL() 2/2

```
String QuantConnect.Algorithm.QCAlgorithm.SEDOL (
Symbol symbol
)
```

Converts a `Symbol` into a SEDOL identifier.

Show Details

Parameters		
`Symbol`	symbol	The `Symbol` .

Return

`String` - SEDOL corresponding to the Symbol. If no matching SEDOL is found, returns None.

Definition at line 2996 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

Sell() 1/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Sell (
 Symbol symbol,
 Int32 quantity
)
```

Sell stock (alias of Order).

Show Details

Parameters		
`Symbol`	symbol	string Symbol of the asset to trade.
`Int32`	quantity	int Quantity of the asset to trade.

Return

`OrderTicket` - The order ticket instance.

Definition at line 102 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Sell() 2/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Sell (
 Symbol symbol,
 Double quantity
)
```

Sell stock (alias of Order).

Show Details

Parameters		
Symbol	symbol	String symbol to sell.
Double	quantity	Quantity to order.

Return

OrderTicket - The order ticket instance.

Definition at line 114 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Sell() 3/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Sell (
 Symbol symbol,
 Single quantity
)
```

Sell stock (alias of Order).

Show Details

Parameters		
Symbol	symbol	String symbol.
Single	quantity	Quantity to sell.

Return

OrderTicket - The order ticket instance.

Definition at line 126 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Sell() 4/5

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.Sell (
Symbol symbol,
Decimal quantity
)
```

Sell stock (alias of Order).

Show Details

Parameters		
Symbol	symbol	String symbol to sell.
Decimal	quantity	Quantity to sell.

Return

OrderTicket - The order ticket instance.

Definition at line 138 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

Sell() 5/5

```
IEnumerable<OrderTicket> QuantConnect.Algorithm.QCAlgorithm.Sell (
OptionStrategy strategy,
Int32 quantity,
*Boolean asynchronous,
*String tag,
*IOrderProperties orderProperties
)
```

Sell Option Strategy (alias of Order).

Show Details

Parameters		
`OptionStrategy`	strategy	Specification of the strategy to trade.
`Int32`	quantity	Quantity of the strategy to trade.
`*Boolean`	asynchronous	*(Optional)* Send the order asynchronously (False). Otherwise we'll block until it fills.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`IEnumerable<OrderTicket>` - Sequence of order tickets.

Definition at line 643 of file Algorithm/QCAlgorithm.Trading.cs.

SECURITIES AND PORTFOLIO

SetAccountCurrency() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetAccountCurrency (
 String accountCurrency
)
```

Sets the account currency cash symbol this algorithm is to manage.

Show Details

Parameters		
`String`	accountCurrency	The account currency cash symbol to set.

Return

`Void` - This method provides no return.

Definition at line 1327 of file Algorithm/QCAlgorithm.cs.

ALGORITHM FRAMEWORK

SetAlpha() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetAlpha (
IAlphaModel alpha
)
```

Sets the alpha model.

Show Details

Parameters		
IAlphaModel	alpha	Model that generates alpha.

Return

Void - This method provides no return.

Definition at line 316 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK

SetAlpha() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetAlpha (
PyObject alpha
)
```

Sets the alpha model.

Show Details

Parameters		
PyObject	alpha	Model that generates alpha.

Return

Void - This method provides no return.

Definition at line 31 of file Algorithm/QCAlgorithm.Framework.Python.cs.

HANDLING DATA

SetApi() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetApi (
IApi api
)
```

Provide the API for the algorithm.

Show Details

Parameters		
IApi	api	Initiated API.

Return

`Void` - This method provides no return.

Definition at line 2818 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS SECURITIES AND PORTFOLIO INDICATORS

SetBenchmark() 1/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBenchmark (
PyObject benchmark
)
```

Sets the specified function as the benchmark, this function provides the value of the benchmark at each date/time requested.

Show Details

Parameters		
PyObject	benchmark	The benchmark producing function.

Return

Void - This method provides no return.

Definition at line 1101 of file Algorithm/QCAlgorithm.Python.cs.

TRADING AND ORDERS | SECURITIES AND PORTFOLIO | INDICATORS

SetBenchmark() 2/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBenchmark (
 SecurityType securityType,
 String symbol
)
```

Sets the benchmark used for computing statistics of the algorithm to the specified symbol.

Show Details

Parameters		
SecurityType	securityType	Is the symbol an equity, forex, base, etc. Default SecurityType.Equity.
String	symbol	symbol to use as the benchmark.

Return

Void - This method provides no return.

Definition at line 1218 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS | SECURITIES AND PORTFOLIO | INDICATORS

SetBenchmark() 3/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBenchmark (
 String ticker
)
```

Sets the benchmark used for computing statistics of the algorithm to the specified ticker, defaulting to SecurityType.Equity if the ticker doesn't exist in the algorithm.

Show Details

Parameters		
String	ticker	Ticker to use as the benchmark.

Return

Void - This method provides no return.

Definition at line 1246 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS SECURITIES AND PORTFOLIO INDICATORS

SetBenchmark() 4/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBenchmark (
 Symbol symbol
)
```

Sets the benchmark used for computing statistics of the algorithm to the specified symbol.

Show Details

Parameters		
Symbol	symbol	symbol to use as the benchmark.

Return

Void - This method provides no return.

Definition at line 1275 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS SECURITIES AND PORTFOLIO INDICATORS

SetBenchmark() 5/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBenchmark (
Func<DateTime, Decimal> benchmark
)
```

Sets the specified function as the benchmark, this function provides the value of the benchmark at each date/time requested.

Show Details

Parameters		
`Func<DateTime, Decimal>`	benchmark	The benchmark producing function.

Return

`Void` - This method provides no return.

Definition at line 1294 of file Algorithm/QCAlgorithm.cs.

MODELING LOGGING

SetBrokerageMessageHandler() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBrokerageMessageHandler (
IBrokerageMessageHandler handler
)
```

Sets the implementation used to handle messages from the brokerage. The default implementation will forward messages to debug or error and when a `Error` occurs, the algorithm is stopped.

Show Details

Parameters		
`IBrokerageMessageHandler`	handler	The message handler to use.

Return

`Void` - This method provides no return.

Definition at line 1202 of file Algorithm/QCAlgorithm.cs.

MODELING

SetBrokerageModel() 1/3

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBrokerageModel (
PyObject model
)
```

Sets the brokerage to emulate in backtesting or paper trading. This can be used to set a custom brokerage model.

Show Details

Parameters		
PyObject	model	The brokerage model to use.

Return

Void - This method provides no return.

Definition at line 1123 of file Algorithm/QCAlgorithm.Python.cs.

MODELING

SetBrokerageModel() 2/3

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBrokerageModel (
BrokerageName brokerage,
*AccountType accountType
)
```

Sets the brokerage to emulate in backtesting or paper trading. This can be used for brokerages that have been implemented in LEAN.

Show Details

Parameters		
BrokerageName	brokerage	The brokerage to emulate. *Options: ['Default', 'QuantConnectBrokerage', 'InteractiveBrokersBrokerage', 'TradierBrokerage', 'OandaBrokerage', 'FxcmBrokerage', 'Bitfinex', 'Binance', 'GDAX', 'Alpaca', 'AlphaStreams', 'Zerodha', 'Samco', 'Atreyu', 'TradingTechnologies', 'Kraken', 'FTX', 'FTXUS', 'Exante', 'BinanceUS', 'Wolverine', 'TDAmeritrade', 'BinanceFutures', 'BinanceCoinFutures', 'RBI']*
*AccountType	accountType	*(Optional)* The account type (Cash or Margin). *Options: ['Margin', 'Cash']*

Return

Void - This method provides no return.

Definition at line 1157 of file Algorithm/QCAlgorithm.cs.

MODELING

SetBrokerageModel() 3/3

```
Void QuantConnect.Algorithm.QCAlgorithm.SetBrokerageModel (
 IBrokerageModel model
)
```

Sets the brokerage to emulate in backtesting or paper trading. This can be used to set a custom brokerage model.

Show Details

Parameters		
IBrokerageModel	model	The brokerage model to use.

Return

Void - This method provides no return.

Definition at line 1168 of file Algorithm/QCAlgorithm.cs.

SECURITIES AND PORTFOLIO

SetCash() 1/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetCash (
Int32 startingCash
)
```

Set initial cash for the strategy while backtesting. During live mode this value is ignored and replaced with the actual cash of your brokerage account.

Show Details

Parameters		
Int32	startingCash	Starting cash for the strategy backtest.

Return

Void - This method provides no return.

Definition at line 1359 of file Algorithm/QCAlgorithm.cs.

SECURITIES AND PORTFOLIO

SetCash() 2/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetCash (
Decimal startingCash
)
```

Set initial cash for the strategy while backtesting. During live mode this value is ignored and replaced with the actual cash of your brokerage account.

Show Details

Parameters		
Decimal	startingCash	Starting cash for the strategy backtest.

Return

Void - This method provides no return.

Definition at line 1370 of file Algorithm/QCAlgorithm.cs.

SECURITIES AND PORTFOLIO

SetCash() 3/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetCash (
String symbol,
Decimal startingCash,
*Decimal conversionRate
)
```

Set the cash for the specified symbol.

Show Details

Parameters		
String	symbol	The cash symbol to set.
Decimal	startingCash	Decimal cash value of portfolio.
*Decimal	conversionRate	*(Optional)* The current conversion rate for the.

Return

Void - This method provides no return.

Definition at line 1389 of file Algorithm/QCAlgorithm.cs.

SECURITIES AND PORTFOLIO

SetCash() 4/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetCash (
Double startingCash
)
```

Set initial cash for the strategy while backtesting. During live mode this value is ignored and replaced with the actual cash of your brokerage account.

Show Details

Parameters		
Double	startingCash	Starting cash for the strategy backtest.

Return

Void - This method provides no return.

Definition at line 1347 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetCurrentSlice() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetCurrentSlice (
Slice slice
)
```

Sets the current slice.

Show Details

Parameters		
Slice	slice	The Slice object.

Return

Void - This method provides no return.

Definition at line 2807 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetEndDate() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetEndDate (
Int32 year,
Int32 month,
Int32 day
)
```

Set the end date for a backtest run.

Show Details

Parameters		
Int32	year	Int year end date.
Int32	month	Int month end date.
Int32	day	Int end date 1-30.

Return

Void - This method provides no return.

Definition at line 1438 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetEndDate() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetEndDate (
DateTime end
)
```

Set the end date for a backtest.

Show Details

Parameters		
DateTime	end	Datetime value for end date.

Return

Void - This method provides no return.

Definition at line 1514 of file Algorithm/QCAlgorithm.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

SetExecution() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetExecution (
IExecutionModel execution
)
```

Sets the execution model.

Show Details

Parameters		
IExecutionModel	execution	Model defining how to execute trades to reach a portfolio target.

Return

Void - This method provides no return.

Definition at line 362 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

SetExecution() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetExecution (
PyObject execution
)
```

Sets the execution model.

Show Details

Parameters		
PyObject	execution	Model defining how to execute trades to reach a portfolio target.

Return

Void - This method provides no return.

Definition at line 67 of file Algorithm/QCAlgorithm.Framework.Python.cs.

ADDING DATA

SetFutureChainProvider() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetFutureChainProvider (
IFutureChainProvider futureChainProvider
)
```

Sets the future chain provider, used to get the list of future contracts for an underlying symbol.

Show Details

Parameters		
IFutureChainProvider	futureChainProvider	The future chain provider.

Return

Void - This method provides no return.

Definition at line 837 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

SetHoldings() 1/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetHoldings (
List<PortfolioTarget> targets,
*Boolean liquidateExistingHoldings,
*String tag,
*IOrderProperties orderProperties
)
```

Sets holdings for a collection of targets. The implementation will order the provided targets executing first those that reduce a position, freeing margin.

Show Details

Parameters		
`List<PortfolioTarget>`	targets	The portfolio desired quantities as percentages.
`*Boolean`	liquidateExistingHoldings	*(Optional)* True will liquidate existing holdings.
`*String`	tag	*(Optional)* Tag the order with a short string.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`Void` - This method provides no return.

Definition at line 1155 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

SetHoldings() 2/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetHoldings (
Symbol symbol,
Double percentage,
*Boolean liquidateExistingHoldings,
*String tag,
*IOrderProperties orderProperties
)
```

Alias for SetHoldings to avoid the M-decimal errors.

Show Details

Parameters		
`Symbol`	symbol	string symbol we wish to hold.
`Double`	percentage	double percentage of holdings desired.
`*Boolean`	liquidateExistingHoldings	*(Optional)* liquidate existing holdings if necessary to hold this stock.
`*String`	tag	*(Optional)* Tag the order with a short string.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`Void` - This method provides no return.

Definition at line 1182 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

SetHoldings() 3/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetHoldings (
Symbol symbol,
Single percentage,
*Boolean liquidateExistingHoldings,
*String tag,
*IOrderProperties orderProperties
)
```

Alias for SetHoldings to avoid the M-decimal errors.

Show Details

Parameters		
`Symbol`	symbol	string symbol we wish to hold.
`Single`	percentage	float percentage of holdings desired.
`*Boolean`	liquidateExistingHoldings	*(Optional)* bool liquidate existing holdings if necessary to hold this stock.
`*String`	tag	*(Optional)* Tag the order with a short string.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`Void` - This method provides no return.

Definition at line 1197 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

SetHoldings() 4/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetHoldings (
Symbol symbol,
Int32 percentage,
*Boolean liquidateExistingHoldings,
*String tag,
*IOrderProperties orderProperties
)
```

Alias for SetHoldings to avoid the M-decimal errors.

Show Details

Parameters		
`Symbol`	symbol	string symbol we wish to hold.
`Int32`	percentage	float percentage of holdings desired.
`*Boolean`	liquidateExistingHoldings	*(Optional)* bool liquidate existing holdings if necessary to hold this stock.
`*String`	tag	*(Optional)* Tag the order with a short string.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`Void` - This method provides no return.

Definition at line 1212 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

SetHoldings() 5/5

```
Void QuantConnect.Algorithm.QCAlgorithm.SetHoldings (
Symbol symbol,
Decimal percentage,
*Boolean liquidateExistingHoldings,
*String tag,
*IOrderProperties orderProperties
)
```

Automatically place a market order which will set the holdings to between 100% or -100% of *PORTFOLIO VALUE*. E.g. SetHoldings("AAPL", 0.1); SetHoldings("IBM", -0.2); -> Sets portfolio as long 10% APPL and short 20% IBM E.g. SetHoldings("AAPL", 2); -> Sets apple to 2x leveraged with all our cash. If the market is closed, place a market on open order.

Show Details

Parameters		
`Symbol`	symbol	Symbol indexer.
`Decimal`	percentage	decimal fraction of portfolio to set stock.
`*Boolean`	liquidateExistingHoldings	*(Optional)* bool flag to clean all existing holdings before setting new faction.
`*String`	tag	*(Optional)* Tag the order with a short string.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`Void` - This method provides no return.

Definition at line 1230 of file Algorithm/QCAlgorithm.Trading.cs.

HANDLING DATA MACHINE LEARNING

SetObjectStore() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetObjectStore (
IObjectStore objectStore
)
```

Sets the object store.

Show Details

Parameters		
`IObjectStore`	objectStore	The object store.

Return

`Void` - This method provides no return.

Definition at line 2828 of file Algorithm/QCAlgorithm.cs.

ADDING DATA

SetOptionChainProvider() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetOptionChainProvider (
IOptionChainProvider optionChainProvider
)
```

Sets the option chain provider, used to get the list of option contracts for an underlying symbol.

Show Details

Parameters		
IOptionChainProvider	optionChainProvider	The option chain provider.

Return

Void - This method provides no return.

Definition at line 827 of file Algorithm/QCAlgorithm.cs.

<-- Missing documentation attribute for SetPandasConverter -->

PARAMETER AND OPTIMIZATION

SetParameters() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetParameters (
Dictionary<String, String> parameters
)
```

Show Details

Parameters		
Dictionary<String, String>	parameters	/

Return

Void - This method provides no return.

Definition at line 740 of file Algorithm/QCAlgorithm.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

SetPortfolioConstruction() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetPortfolioConstruction (
IPortfolioConstructionModel portfolioConstruction
)
```

Sets the portfolio construction model.

Show Details

Parameters		
IPortfolioConstructionModel	portfolioConstruction	Model defining how to build a portfolio from insights.

Return

Void - This method provides no return.

Definition at line 351 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

SetPortfolioConstruction() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetPortfolioConstruction (
PyObject portfolioConstruction
)
```

Sets the portfolio construction model.

Show Details

Parameters		
PyObject	portfolioConstruction	Model defining how to build a portfolio from alphas.

Return

Void - This method provides no return.

Definition at line 86 of file Algorithm/QCAlgorithm.Framework.Python.cs.

LOGGING

SetQuit() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetQuit (
Boolean quit
)
```

Set the Quit flag property of the algorithm.

Show Details

Parameters		
Boolean	quit	Boolean quit state.

Return

Void - This method provides no return.

Definition at line 2628 of file Algorithm/QCAlgorithm.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

SetRiskManagement() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRiskManagement (
IRiskManagementModel riskManagement
)
```

Sets the risk management model.

Show Details

Parameters		
IRiskManagementModel	riskManagement	Model defining how risk is managed.

Return

Void - This method provides no return.

Definition at line 373 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK TRADING AND ORDERS

SetRiskManagement() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRiskManagement (
 PyObject riskManagement
)
```

Sets the risk management model.

Show Details

Parameters		
PyObject	riskManagement	Model defining how risk is managed.

Return

Void - This method provides no return.

Definition at line 137 of file Algorithm/QCAlgorithm.Framework.Python.cs.

HANDLING DATA LIVE TRADING

SetRunTimeError() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRunTimeError (
Exception exception
)
```

Set the runtime error.

Show Details

Parameters		
Exception	exception	Represents error that occur during execution.

Return

Void - This method provides no return.

Definition at line 2701 of file Algorithm/QCAlgorithm.cs.

CHARTING

SetRuntimeStatistic() 1/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRuntimeStatistic (
String name,
String value
)
```

Set a runtime statistic for the algorithm. Runtime statistics are shown in the top banner of a live algorithm GUI.

Show Details

Parameters		
String	name	Name of your runtime statistic.
String	value	String value of your runtime statistic.

Return

Void - This method provides no return.

Definition at line 297 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

SetRuntimeStatistic() 2/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRuntimeStatistic (
String name,
Decimal value
)
```

Set a runtime statistic for the algorithm. Runtime statistics are shown in the top banner of a live algorithm GUI.

Show Details

Parameters		
String	name	Name of your runtime statistic.
Decimal	value	Decimal value of your runtime statistic.

Return

Void - This method provides no return.

Definition at line 308 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

SetRuntimeStatistic() 3/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRuntimeStatistic (
String name,
Int32 value
)
```

Set a runtime statistic for the algorithm. Runtime statistics are shown in the top banner of a live algorithm GUI.

Show Details

Parameters		
String	name	Name of your runtime statistic.
Int32	value	Int value of your runtime statistic.

Return

Void - This method provides no return.

Definition at line 319 of file Algorithm/QCAlgorithm.Plotting.cs.

CHARTING

SetRuntimeStatistic() 4/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetRuntimeStatistic (
String name,
Double value
)
```

Set a runtime statistic for the algorithm. Runtime statistics are shown in the top banner of a live algorithm GUI.

Show Details

Parameters		
String	name	Name of your runtime statistic.
Double	value	Double value of your runtime statistic.

Return

Void - This method provides no return.

Definition at line 330 of file Algorithm/QCAlgorithm.Plotting.cs.

ADDING DATA MODELING

SetSecurityInitializer() 1/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetSecurityInitializer (
PyObject securityInitializer
)
```

Sets the security initializer function, used to initialize/configure securities after creation.

Show Details

Parameters		
PyObject	securityInitializer	The security initializer function or class.

Return

Void - This method provides no return.

Definition at line 1139 of file Algorithm/QCAlgorithm.Python.cs.

ADDING DATA MODELING

SetSecurityInitializer() 2/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetSecurityInitializer (
ISecurityInitializer securityInitializer
)
```

Sets the security initializer, used to initialize/configure securities after creation. The initializer will be applied to all universes and manually added securities.

Show Details

Parameters		
ISecurityInitializer	securityInitializer	The security initializer.

Return

Void - This method provides no return.

Definition at line 778 of file Algorithm/QCAlgorithm.cs.

ADDING DATA MODELING

SetSecurityInitializer() 3/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetSecurityInitializer (
Action<Security, Boolean> securityInitializer
)
```

Show Details

Parameters		
Action<Security, Boolean>	securityInitializer	/

Return

Void - This method provides no return.

Definition at line 804 of file Algorithm/QCAlgorithm.cs.

ADDING DATA MODELING

SetSecurityInitializer() 4/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetSecurityInitializer (
Action<Security> securityInitializer
)
```

Show Details

Parameters		
Action<Security>	securityInitializer	/

Return

Void - This method provides no return.

Definition at line 816 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetStartDate() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetStartDate (
 Int32 year,
 Int32 month,
 Int32 day
)
```

Set the start date for backtest.

Show Details

Parameters		
Int32	year	Int year starting date.
Int32	month	Int month starting date.
Int32	day	Int starting date 1-30.

Return

Void - This method provides no return.

Definition at line 1412 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetStartDate() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetStartDate (
 DateTime start
)
```

Set the start date for the backtest.

Show Details

Parameters		
DateTime	start	Datetime Start date for backtest.

Return

Void - This method provides no return.

Definition at line 1473 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetTimeZone() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetTimeZone (
String timeZone
)
```

Sets the time zone of the Time property in the algorithm.

Show Details

Parameters		
String	timeZone	The desired time zone.

Return

Void - This method provides no return.

Definition at line 1099 of file Algorithm/QCAlgorithm.cs.

HANDLING DATA

SetTimeZone() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetTimeZone (
DateTimeZone timeZone
)
```

Sets the time zone of the `Time` property in the algorithm.

Show Details

Parameters		
`DateTimeZone`	timeZone	The desired time zone.

Return

`Void` - This method provides no return.

Definition at line 1119 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

SetTradeBuilder() 1/1

```
Void QuantConnect.Algorithm.QCAlgorithm.SetTradeBuilder (
 ITradeBuilder tradeBuilder
)
```

Set the `ITradeBuilder` implementation to generate trades from executions and market price updates.

Show Details

Parameters		
`ITradeBuilder`	tradeBuilder	/

Return

`Void` - This method provides no return.

Definition at line 1580 of file Algorithm/QCAlgorithm.cs.

ALGORITHM FRAMEWORK UNIVERSES

SetUniverseSelection() 1/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetUniverseSelection (
IUniverseSelectionModel universeSelection
)
```

Sets the universe selection model.

Show Details

Parameters		
IUniverseSelectionModel	universeSelection	Model defining universes for the algorithm.

Return

Void - This method provides no return.

Definition at line 279 of file Algorithm/QCAlgorithm.Framework.cs.

ALGORITHM FRAMEWORK UNIVERSES

SetUniverseSelection() 2/2

```
Void QuantConnect.Algorithm.QCAlgorithm.SetUniverseSelection (
PyObject universeSelection
)
```

Sets the universe selection model.

Show Details

Parameters		
PyObject	universeSelection	Model defining universes for the algorithm.

Return

Void - This method provides no return.

Definition at line 105 of file Algorithm/QCAlgorithm.Framework.Python.cs.

HISTORICAL DATA

SetWarmUp() 1/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetWarmUp (
TimeSpan timeSpan
)
```

Sets the warm up period to the specified value.

Show Details

Parameters		
TimeSpan	timeSpan	The amount of time to warm up, this does not take into account market hours/weekends.

Return

Void - This method provides no return.

Definition at line 65 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

SetWarmUp() 2/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetWarmUp (
TimeSpan timeSpan,
Nullable<Resolution> resolution
)
```

Sets the warm up period to the specified value.

Show Details

Parameters		
TimeSpan	timeSpan	The amount of time to warm up, this does not take into account market hours/weekends.
Nullable<Resolution>	resolution	The resolution to request.

Return

Void - This method provides no return.

Definition at line 87 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

SetWarmUp() 3/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetWarmUp (
Int32 barCount
)
```

Sets the warm up period by resolving a start date that would send that amount of data into the algorithm. The highest (smallest) resolution in the securities collection will be used. For example, if an algorithm has minute and daily data and 200 bars are requested, that would use 200 minute bars.

Show Details

Parameters		
Int32	barCount	The number of data points requested for warm up.

Return

Void - This method provides no return.

Definition at line 113 of file Algorithm/QCAlgorithm.History.cs.

HISTORICAL DATA

SetWarmUp() 4/4

```
Void QuantConnect.Algorithm.QCAlgorithm.SetWarmUp (
Int32 barCount,
Nullable<Resolution> resolution
)
```

Sets the warm up period by resolving a start date that would send that amount of data into the algorithm.

Show Details

Parameters		
Int32	barCount	The number of data points requested for warm up.
Nullable<Resolution>	resolution	The resolution to request.

Return

Void - This method provides no return.

Definition at line 137 of file Algorithm/QCAlgorithm.History.cs.

TRADING AND ORDERS

Shortable() 1/2

```
Boolean QuantConnect.Algorithm.QCAlgorithm.Shortable (
Symbol symbol
)
```

Determines if the Symbol is shortable at the brokerage.

Show Details

Parameters		
Symbol	symbol	Symbol to check if shortable.

Return

Boolean - True if shortable.

Definition at line 2840 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

Shortable() 2/2

```
Boolean QuantConnect.Algorithm.QCAlgorithm.Shortable (
 Symbol symbol,
 Decimal shortQuantity
)
```

Determines if the Symbol is shortable at the brokerage.

Show Details

Parameters		
Symbol	symbol	Symbol to check if shortable.
Decimal	shortQuantity	Order's quantity to check if it is currently shortable, taking into account current holdings and open orders.

Return

Boolean - True if shortable.

Definition at line 2852 of file Algorithm/QCAlgorithm.cs.

TRADING AND ORDERS

ShortableQuantity() 1/1

```
Int64 QuantConnect.Algorithm.QCAlgorithm.ShortableQuantity (
 Symbol symbol
)
```

Gets the quantity shortable for the given asset.

Show Details

Parameters		
`Symbol`	symbol	/

Return

`Int64` - Quantity shortable for the given asset. Zero if not shortable, or a number greater than zero if shortable.

Definition at line 2881 of file Algorithm/QCAlgorithm.cs.

INDICATORS

SI() 1/1

```
WilderSwingIndex QuantConnect.Algorithm.QCAlgorithm.SI (
Symbol symbol,
Decimal limitMove,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a Wilder Swing Index (SI) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose SI we want.
`Decimal`	limitMove	The maximum daily change in price for the SI.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`WilderSwingIndex` - The WilderSwingIndex for the given parameters.

Definition at line 1906 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

SMA() 1/1

```
SimpleMovingAverage QuantConnect.Algorithm.QCAlgorithm.SMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates an SimpleMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose SMA we want.
`Int32`	period	The period of the SMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SimpleMovingAverage` - The SimpleMovingAverage for the given parameters.

Definition at line 1528 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

SORTINO() 1/1

```
SortinoRatio QuantConnect.Algorithm.QCAlgorithm.SORTINO (
Symbol symbol,
Int32 sortinoPeriod,
*Double minimumAcceptableReturn,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Sortino indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Sortino we want.
`Int32`	sortinoPeriod	Period of historical observation for Sortino ratio calculation.
`*Double`	minimumAcceptableReturn	*(Optional)* Minimum acceptable return (eg risk-free rate) for the Sortino ratio calculation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SortinoRatio` - The SortinoRatio indicator for the requested symbol over the specified period.

Definition at line 1508 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

SR() 1/1

```
SharpeRatio QuantConnect.Algorithm.QCAlgorithm.SR (
Symbol symbol,
Int32 sharpePeriod,
*Decimal riskFreeRate,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new RollingSharpeRatio indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose RSR we want.
Int32	sharpePeriod	Period of historical observation for sharpe ratio calculation.
*Decimal	riskFreeRate	*(Optional)* Risk-free rate for sharpe ratio calculation.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

SharpeRatio - The RollingSharpeRatio indicator for the requested symbol over the specified period.

Definition at line 1489 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

STC() 1/1

```
SchaffTrendCycle QuantConnect.Algorithm.QCAlgorithm.STC (
 Symbol symbol,
 Int32 cyclePeriod,
 Int32 fastPeriod,
 Int32 slowPeriod,
 *MovingAverageType movingAverageType,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new Schaff Trend Cycle indicator.

Show Details

Parameters		
Symbol	symbol	The symbol for the indicator to track.
Int32	cyclePeriod	The signal period.
Int32	fastPeriod	The fast moving average period.
Int32	slowPeriod	The slow moving average period.
*MovingAverageType	movingAverageType	*(Optional)* The type of moving average to use.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

SchaffTrendCycle - The SchaffTrendCycle indicator for the requested symbol over the specified period.

Definition at line 1549 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

STD() 1/1

```
StandardDeviation QuantConnect.Algorithm.QCAlgorithm.STD (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a new StandardDeviation indicator. This will return the population standard deviation of samples over the specified period.

Show Details

Parameters		
Symbol	symbol	The symbol whose STD we want.
Int32	period	The period over which to compute the STD.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

StandardDeviation - The StandardDeviation indicator for the requested symbol over the specified period.

Definition at line 1567 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

STO() 1/2

```
Stochastic QuantConnect.Algorithm.QCAlgorithm.STO (
Symbol symbol,
Int32 period,
Int32 kPeriod,
Int32 dPeriod,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Stochastic indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose stochastic we seek.
`Int32`	period	The period of the stochastic. Normally 14.
`Int32`	kPeriod	The sum period of the stochastic. Normally 14.
`Int32`	dPeriod	The sum period of the stochastic. Normally 3.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`Stochastic` - Stochastic indicator for the requested symbol.

Definition at line 1607 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

STO() 2/2

```
Stochastic QuantConnect.Algorithm.QCAlgorithm.STO (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, TradeBar> selector
)
```

Overload short hand to create a new Stochastic indicator; defaulting to the 3 period for dStoch.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose stochastic we seek.
`Int32`	period	The period of the stochastic. Normally 14.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`Stochastic` - Stochastic indicator for the requested symbol.

Definition at line 1626 of file Algorithm/QCAlgorithm.Indicators.cs.

TRADING AND ORDERS

StopLimitOrder() 1/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.StopLimitOrder (
Symbol symbol,
Int32 quantity,
Decimal stopPrice,
Decimal limitPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Send a stop limit order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Int32`	quantity	Quantity of shares for limit order.
`Decimal`	stopPrice	Stop price for this order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 488 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

StopLimitOrder() 2/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.StopLimitOrder (
 Symbol symbol,
 Double quantity,
 Decimal stopPrice,
 Decimal limitPrice,
 *String tag,
 *IOrderProperties orderProperties
 )
```

Send a stop limit order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Double`	quantity	Quantity of shares for limit order.
`Decimal`	stopPrice	Stop price for this order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 504 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

StopLimitOrder() 3/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.StopLimitOrder (
 Symbol symbol,
 Decimal quantity,
 Decimal stopPrice,
 Decimal limitPrice,
 *String tag,
 *IOrderProperties orderProperties
 )
```

Send a stop limit order to the transaction handler:.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset.
`Decimal`	quantity	Quantity of shares for limit order.
`Decimal`	stopPrice	Stop price for this order.
`Decimal`	limitPrice	Limit price to fill this order.
`*String`	tag	*(Optional)* String tag for the order (optional).
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 520 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

StopMarketOrder() 1/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.StopMarketOrder (
Symbol symbol,
Int32 quantity,
Decimal stopPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Create a stop market order and return the newly created order id; or negative if the order is invalid.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset we're trading.
`Int32`	quantity	Quantity to be traded.
`Decimal`	stopPrice	Price to fill the stop order.
`*String`	tag	*(Optional)* Optional string data tag for the order.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 439 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

StopMarketOrder() 2/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.StopMarketOrder (
Symbol symbol,
Double quantity,
Decimal stopPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Create a stop market order and return the newly created order id; or negative if the order is invalid.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset we're trading.
`Double`	quantity	Quantity to be traded.
`Decimal`	stopPrice	Price to fill the stop order.
`*String`	tag	*(Optional)* Optional string data tag for the order.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 454 of file Algorithm/QCAlgorithm.Trading.cs.

TRADING AND ORDERS

StopMarketOrder() 3/3

```
OrderTicket QuantConnect.Algorithm.QCAlgorithm.StopMarketOrder (
Symbol symbol,
Decimal quantity,
Decimal stopPrice,
*String tag,
*IOrderProperties orderProperties
)
```

Create a stop market order and return the newly created order id; or negative if the order is invalid.

Show Details

Parameters		
`Symbol`	symbol	String symbol for the asset we're trading.
`Decimal`	quantity	Quantity to be traded.
`Decimal`	stopPrice	Price to fill the stop order.
`*String`	tag	*(Optional)* Optional string data tag for the order.
`*IOrderProperties`	orderProperties	*(Optional)* The order properties to use. Defaults to `DefaultOrderProperties` .

Return

`OrderTicket` - The order ticket instance.

Definition at line 469 of file Algorithm/QCAlgorithm.Trading.cs.

INDICATORS

STR() 1/1

```
SuperTrend QuantConnect.Algorithm.QCAlgorithm.STR (
Symbol symbol,
Int32 period,
Decimal multiplier,
*MovingAverageType movingAverageType,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new SuperTrend indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose SuperTrend indicator we want.
`Int32`	period	The smoothing period for average True range.
`Decimal`	multiplier	Multiplier to calculate basic upper and lower bands width.
`*MovingAverageType`	movingAverageType	*(Optional)* Smoother type for average True range, defaults to Wilders.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`SuperTrend` - The new `SuperTrend` object.

Definition at line 1469 of file Algorithm/QCAlgorithm.Indicators.cs.

<-- Missing documentation attribute for SubmitOrderRequest -->

INDICATORS

SUM() 1/1

```
Sum QuantConnect.Algorithm.QCAlgorithm.SUM (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Sum indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Sum we want.
`Int32`	period	The period over which to compute the Sum.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Sum` - The Sum indicator for the requested symbol over the specified period.

Definition at line 1640 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

SWISS() 1/1

```
SwissArmyKnife QuantConnect.Algorithm.QCAlgorithm.SWISS (
Symbol symbol,
Int32 period,
Double delta,
SwissArmyKnifeTool tool,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates Swiss Army Knife transformation for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol to use for calculations.
`Int32`	period	The period of the calculation.
`Double`	delta	The delta scale of the BandStop or BandPass.
`SwissArmyKnifeTool`	tool	The tool os the Swiss Army Knife. *Options: ['Gauss', 'Butter', 'HighPass', 'TwoPoleHighPass', 'BandPass']*
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* elects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`SwissArmyKnife` - The calculation using the given tool.

Definition at line 1661 of file Algorithm/QCAlgorithm.Indicators.cs.

ADDING DATA HANDLING DATA

Symbol() 1/1

```
Symbol QuantConnect.Algorithm.QCAlgorithm.Symbol (
 String ticker
)
```

Converts the string 'ticker' symbol into a full `Symbol` object This requires that the string 'ticker' has been added to the algorithm.

Show Details

Parameters		
`String`	ticker	The ticker symbol. This should be the ticker symbol as it was added to the algorithm.

Return

`Symbol` - The symbol object mapped to the specified ticker.

Definition at line 2644 of file Algorithm/QCAlgorithm.cs.

INDICATORS

T3() 1/1

```
T3MovingAverage QuantConnect.Algorithm.QCAlgorithm.T3 (
Symbol symbol,
Int32 period,
*Decimal volumeFactor,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new T3MovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose T3 we want.
`Int32`	period	The period over which to compute the T3.
`*Decimal`	volumeFactor	*(Optional)* The volume factor to be used for the T3 (value must be in the [0,1] range, defaults to 0.7).
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`T3MovingAverage` - The T3MovingAverage indicator for the requested symbol over the specified period.

Definition at line 1680 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TDD() 1/1

```
TargetDownsideDeviation QuantConnect.Algorithm.QCAlgorithm.TDD (
Symbol symbol,
Int32 period,
*Double minimumAcceptableReturn,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new TargetDownsideDeviation indicator. The target downside deviation is defined as the root-mean-square, or RMS, of the deviations of the realized return's underperformance from the target return where all returns above the target return are treated as underperformance of 0.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TDD we want.
`Int32`	period	The period over which to compute the TDD.
`*Double`	minimumAcceptableReturn	*(Optional)* Minimum acceptable return (MAR) for the target downside deviation calculation.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TargetDownsideDeviation` - The TargetDownsideDeviation indicator for the requested symbol over the specified period.

Definition at line 1587 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TEMA() 1/1

```
TripleExponentialMovingAverage QuantConnect.Algorithm.QCAlgorithm.TEMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new TripleExponentialMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TEMA we want.
`Int32`	period	The period over which to compute the TEMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TripleExponentialMovingAverage` - The TripleExponentialMovingAverage indicator for the requested symbol over the specified period.

Definition at line 1698 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TP() 1/1

```
TimeProfile QuantConnect.Algorithm.QCAlgorithm.TP (
Symbol symbol,
*Int32 period,
*Decimal valueAreaVolumePercentage,
*Decimal priceRangeRoundOff,
*Resolution resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an Market Profile indicator for the symbol with Time Price Opportunity (TPO) mode. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TP we want.
`*Int32`	period	*(Optional)* The period of the TP.
`*Decimal`	valueAreaVolumePercentage	*(Optional)* The percentage of volume contained in the value area.
`*Decimal`	priceRangeRoundOff	*(Optional)* How many digits you want to round and the precision. i.e 0.01 round to two digits exactly.
`*Resolution`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`TimeProfile` - The Time Profile indicator for the given parameters.

Definition at line 998 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TR() 1/1

```
TrueRange QuantConnect.Algorithm.QCAlgorithm.TR (
Symbol symbol,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new TrueRange indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TR we want.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`TrueRange` - The TrueRange indicator for the requested symbol.

Definition at line 1738 of file Algorithm/QCAlgorithm.Indicators.cs.

MACHINE LEARNING SCHEDULED EVENTS

Train() 1/4

```
ScheduledEvent QuantConnect.Algorithm.QCAlgorithm.Train (
 PyObject trainingCode
)
```

Schedules the provided training code to execute immediately.

Show Details

Parameters		
`PyObject`	trainingCode	The training code to be invoked.

Return

`ScheduledEvent` - The new `ScheduledEvent` object.

Definition at line 1334 of file Algorithm/QCAlgorithm.Python.cs.

MACHINE LEARNING SCHEDULED EVENTS

Train() 2/4

```
ScheduledEvent QuantConnect.Algorithm.QCAlgorithm.Train (
IDateRule dateRule,
ITimeRule timeRule,
PyObject trainingCode
)
```

Schedules the training code to run using the specified date and time rules.

Show Details

Parameters		
IDateRule	dateRule	Specifies what dates the event should run.
ITimeRule	timeRule	Specifies the times on those dates the event should run.
PyObject	trainingCode	The training code to be invoked.

Return

ScheduledEvent - The new ScheduledEvent object.

Definition at line 1347 of file Algorithm/QCAlgorithm.Python.cs.

MACHINE LEARNING SCHEDULED EVENTS

Train() 3/4

```
ScheduledEvent QuantConnect.Algorithm.QCAlgorithm.Train (
Action trainingCode
)
```

Schedules the provided training code to execute immediately.

Show Details

Parameters		
Action	trainingCode	The training code to be invoked.

Return

`ScheduledEvent` - The new `ScheduledEvent` object.

Definition at line 2764 of file Algorithm/QCAlgorithm.cs.

MACHINE LEARNING SCHEDULED EVENTS

Train() 4/4

```
ScheduledEvent QuantConnect.Algorithm.QCAlgorithm.Train (
IDateRule dateRule,
ITimeRule timeRule,
Action trainingCode
)
```

Schedules the training code to run using the specified date and time rules.

Show Details

Parameters		
`IDateRule`	dateRule	Specifies what dates the event should run.
`ITimeRule`	timeRule	Specifies the times on those dates the event should run.
`Action`	trainingCode	The training code to be invoked.

Return

`ScheduledEvent` - The new `ScheduledEvent` object.

Definition at line 2777 of file Algorithm/QCAlgorithm.cs.

INDICATORS

TRIMA() 1/1

```
TriangularMovingAverage QuantConnect.Algorithm.QCAlgorithm.TRIMA (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new TriangularMovingAverage indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose TRIMA we want.
`Int32`	period	The period over which to compute the TRIMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`TriangularMovingAverage` - The TriangularMovingAverage indicator for the requested symbol over the specified period.

Definition at line 1756 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TRIN() 1/2

```
ArmsIndex QuantConnect.Algorithm.QCAlgorithm.TRIN (
IEnumerable<Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Arms Index indicator.

Show Details

Parameters		
`IEnumerable<Symbol>`	symbols	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* /

Return

`ArmsIndex` - The new `ArmsIndex` object.

Definition at line 1945 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TRIN() 2/2

```
ArmsIndex QuantConnect.Algorithm.QCAlgorithm.TRIN (
Symbol> symbols,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates a new Arms Index indicator.

Show Details

Parameters		
`Symbol>`	symbols	The symbols whose Arms Index we want.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`ArmsIndex` - The Arms Index indicator for the requested symbol over the specified period.

Definition at line 1958 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TRIX() 1/1

```
Trix QuantConnect.Algorithm.QCAlgorithm.TRIX (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Trix indicator.

Show Details

Parameters		
Symbol	symbol	The symbol whose TRIX we want.
Int32	period	The period over which to compute the TRIX.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

Trix - The Trix indicator for the requested symbol over the specified period.

Definition at line 1774 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

TSI() 1/1

```
TrueStrengthIndex QuantConnect.Algorithm.QCAlgorithm.TSI (
Symbol symbol,
*Int32 longTermPeriod,
*Int32 shortTermPeriod,
*Int32 signalPeriod,
*MovingAverageType signalType,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a TrueStrengthIndex indicator for the symbol. The indicator will be automatically updated on the given

resolution.

Show Details

Parameters		
Symbol	symbol	The symbol whose TSI we want.
*Int32	longTermPeriod	*(Optional)* Period used for the second (double) price change smoothing.
*Int32	shortTermPeriod	*(Optional)* Period used for the first price change smoothing.
*Int32	signalPeriod	*(Optional)* The signal period.
*MovingAverageType	signalType	*(Optional)* The type of moving average to use for the signal.
*Nullable<Resolution>	resolution	*(Optional)* The resolution.
*Func<IBaseData, Decimal>	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

TrueStrengthIndex - The TrueStrengthIndex indicator for the given parameters.

Definition at line 1720 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

ULTOSC() 1/1

```
UltimateOscillator QuantConnect.Algorithm.QCAlgorithm.ULTOSC (
Symbol symbol,
Int32 period1,
Int32 period2,
Int32 period3,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new UltimateOscillator indicator.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose ULTOSC we want.
`Int32`	period1	The first period over which to compute the ULTOSC.
`Int32`	period2	The second period over which to compute the ULTOSC.
`Int32`	period3	The third period over which to compute the ULTOSC.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`UltimateOscillator` - The UltimateOscillator indicator for the requested symbol over the specified period.

Definition at line 1794 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

VAR() 1/1

```
Variance QuantConnect.Algorithm.QCAlgorithm.VAR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Creates a new Variance indicator. This will return the population variance of samples over the specified period.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VAR we want.
`Int32`	period	The period over which to compute the VAR.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`Variance` - The Variance indicator for the requested symbol over the specified period.

Definition at line 1812 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

VP() 1/1

```
VolumeProfile QuantConnect.Algorithm.QCAlgorithm.VP (
Symbol symbol,
*Int32 period,
*Decimal valueAreaVolumePercentage,
*Decimal priceRangeRoundOff,
*Resolution resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an Market Profile indicator for the symbol with Volume Profile (VOL) mode. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VP we want.
`*Int32`	period	*(Optional)* The period of the VP.
`*Decimal`	valueAreaVolumePercentage	*(Optional)* The percentage of volume contained in the value area.
`*Decimal`	priceRangeRoundOff	*(Optional)* How many digits you want to round and the precision. i.e 0.01 round to two digits exactly.
`*Resolution`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`VolumeProfile` - The Volume Profile indicator for the given parameters.

Definition at line 977 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

VWAP() 1/2

```
VolumeWeightedAveragePriceIndicator QuantConnect.Algorithm.QCAlgorithm.VWAP (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, TradeBar> selector
)
```

Creates an VolumeWeightedAveragePrice (VWAP) indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VWAP we want.
`Int32`	period	The period of the VWAP.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, TradeBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`VolumeWeightedAveragePriceIndicator` - The VolumeWeightedAveragePrice for the given parameters.

Definition at line 1831 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

VWAP() 2/2

```
IntradayVwap QuantConnect.Algorithm.QCAlgorithm.VWAP (
 Symbol symbol
)
```

Creates the canonical VWAP indicator that resets each day. The indicator will be automatically updated on the security's configured resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose VWAP we want.

Return

`IntradayVwap` - The IntradayVWAP for the specified symbol.

Definition at line 1847 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS HISTORICAL DATA

WarmUpIndicator() 1/5

```
Void QuantConnect.Algorithm.QCAlgorithm.WarmUpIndicator (
Symbol symbol,
PyObject indicator,
*Nullable<Resolution> resolution,
*PyObject selector
)
```

Warms up a given indicator with historical data.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose indicator we want.
`PyObject`	indicator	The indicator we want to warm up.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*PyObject`	selector	*(Optional)* Selects a value from the BaseData send into the indicator, if None defaults to a cast (x => (T)x).

Return

`Void` - This method provides no return.

Definition at line 659 of file Algorithm/QCAlgorithm.Python.cs.

HISTORICAL DATA INDICATORS

WarmUpIndicator() 2/5

```
Void QuantConnect.Algorithm.QCAlgorithm.WarmUpIndicator (
Symbol symbol,
IndicatorBase<IndicatorDataPoint> indicator,
*Nullable<Resolution> resolution,
*Func<IBaseData, Decimal> selector
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<IndicatorDataPoint>`	indicator	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, Decimal>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2371 of file Algorithm/QCAlgorithm.Indicators.cs.

HISTORICAL DATA INDICATORS

WarmUpIndicator() 3/5

```
Void QuantConnect.Algorithm.QCAlgorithm.WarmUpIndicator (
 Symbol symbol,
 IndicatorBase<IndicatorDataPoint> indicator,
 TimeSpan period,
 *Func<IBaseData, Decimal> selector
 )
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<IndicatorDataPoint>`	indicator	/
`TimeSpan`	period	/
`*Func<IBaseData, Decimal>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2387 of file Algorithm/QCAlgorithm.Indicators.cs.

HISTORICAL DATA INDICATORS

WarmUpIndicator() 4/5

```
Void QuantConnect.Algorithm.QCAlgorithm.WarmUpIndicator (
Symbol symbol,
IndicatorBase<T> indicator,
*Nullable<Resolution> resolution,
*Func<IBaseData, T> selector
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<T>`	indicator	/
`*Nullable<Resolution>`	resolution	*(Optional)* /
`*Func<IBaseData, T>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2413 of file Algorithm/QCAlgorithm.Indicators.cs.

HISTORICAL DATA INDICATORS

WarmUpIndicator() 5/5

```
Void QuantConnect.Algorithm.QCAlgorithm.WarmUpIndicator (
Symbol symbol,
IndicatorBase<T> indicator,
TimeSpan period,
*Func<IBaseData, T> selector
)
```

Show Details

Parameters		
`Symbol`	symbol	/
`IndicatorBase<T>`	indicator	/
`TimeSpan`	period	/
`*Func<IBaseData, T>`	selector	*(Optional)* /

Return

`Void` - This method provides no return.

Definition at line 2430 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

WILR() 1/1

```
WilliamsPercentR QuantConnect.Algorithm.QCAlgorithm.WILR (
Symbol symbol,
Int32 period,
*Nullable<Resolution> resolution,
*Func<IBaseData, IBaseDataBar> selector
)
```

Creates a new Williams %R indicator. This will compute the percentage change of the current closing price in relation to the high and low of the past N periods. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose Williams %R we want.
`Int32`	period	The period over which to compute the Williams %R.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, IBaseDataBar>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to casting the input value to a TradeBar.

Return

`WilliamsPercentR` - The Williams %R indicator for the requested symbol over the specified period.

Definition at line 1866 of file Algorithm/QCAlgorithm.Indicators.cs.

INDICATORS

WWMA() 1/1

```
WilderMovingAverage QuantConnect.Algorithm.QCAlgorithm.WWMA (
 Symbol symbol,
 Int32 period,
 *Nullable<Resolution> resolution,
 *Func<IBaseData, Decimal> selector
)
```

Creates a WilderMovingAverage indicator for the symbol. The indicator will be automatically updated on the given resolution.

Show Details

Parameters		
`Symbol`	symbol	The symbol whose WMA we want.
`Int32`	period	The period of the WMA.
`*Nullable<Resolution>`	resolution	*(Optional)* The resolution.
`*Func<IBaseData, Decimal>`	selector	*(Optional)* Selects a value from the BaseData to send into the indicator, if None defaults to the Value property of BaseData (x => x.Value).

Return

`WilderMovingAverage` - The WilderMovingAverage for the given parameters.

Definition at line 1886 of file Algorithm/QCAlgorithm.Indicators.cs.

Migrations

Migrations

Zipline

The following pages explain how to migrate trading algorithms from Zipline to LEAN:

Initialization

Using Data

Ordering

Logging and Plotting

Quick Reference

Zipline

Initialization

Initializing State

On QuantConnect, algorithm initialization behavior is equivalent to Quantopian's `initialize` method. `Initialize` is the method equivalent, and it is only ran once at the start of the algorithm. It is responsible for initially adding data to the algorithm, adding a universe, setting the brokerage, setting the algorithm's start and end date, etc.

The algorithm structure in QuantConnect differs from Quantopian's as well. On QuantConnect, we create a class that derives from `QCAlgorithm` and define our overrides/algorithm functionality there. An example of an algorithm with initialization is provided.

Quantopian

```py
def initialize(context):
    # Adds AAPL sid to `context` for use later in `handle_data(...)`
    context.aapl = sid(24)
```

QuantConnect

```py
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        # Adds minutely AAPL data to the algorithm
        self.aapl = self.AddEquity("AAPL", Resolution.Minute)
```

On QuantConnect, the `self` parameter is the equivalent of the `context` parameter in Quantopian algorithms. You can use `self` to access algorithm resources, such as the Portfolio, Orders, Order Tickets, Securities, place orders in your algorithm, and much more.

You can read more about algorithm initialization at the Initializion documentation page.

Attaching Pipelines

The functional equivalent of Pipelines on QuantConnect are Universes. Universes can be used to dynamically select assets your algorithm wants to trade or request data from. To add a universe to your algorithm, you can call the `AddUniverse()` method and provide the required Coarse and Fine universe selection functions.

Coarse Universe selection is the first step of universe selection. It provides a means to filter data in a lightweight fashion before proceeding to Fine Universe selection. Coarse universe will allow you to filter assets based off top-level fundamental factors, such as Dollar Volume for the day, if a company has fundamental data, and market of the asset.

Fine Universe selection is the second step of universe selection. It provides fundamental data to your user-defined function, and can be used to filter in greater detail based on an asset's fundamental data, such as earnings, EPS, etc.

An example of coarse and fine universe selection to filter assets in our algorithm is provided. Adding universes can be used as an alternative to using `AddEquity` if you prefer dynamic asset selection.

```
class MyUniverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2020, 1, 1)
        self.SetEndDate(2020, 2, 1)

        self.UniverseSettings.Resolution = Resolution.Minute
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)

    def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        return [i.Symbol for i in coarse if i.DollarVolume > 500000 and i.HasFundamentalData]

    def FineSelectionFunction(self, fine: List[FineFundamental]) -> List[Symbol]:
        return [i.Symbol for i in fine if i.EarningReports.BasicEPS.OneMonth > 0]
```

PY

You can configure universe settings by modifying the existing `UniverseSettings` attribute of the algorithm.

```
class MyUniverseAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.UniverseSettings.Resolution = Resolution.Minute
        self.UniverseSettings.DataNormalizationMode = DataNormalizationMode.Adjusted
        self.UniverseSettings.Leverage = 1.0
        self.UniverseSettings.ExtendedMarketHours = False
```

PY

You can read more about universe selection at the Universe documentation page.

Scheduling Functions

Scheduled Events on QuantConnect are the equivalent of `schedule_function()` from Quantopian. Examples are provided comparing the Scheduled Events functionality between QuantConnect and Quantopian.

Quantopian

```
import quantopian.algorithm as algo

def my_scheduled_function(context, data):
    pass

def initialize(context):
    # Runs a function every day 60 minutes before U.S. Equities market close
    algo.schedule_function(
        func=my_scheduled_function,
        date_rule=algo.date_rules.every_day(),
        time_rule=algo.time_rules.market_close(minutes=60),
        calendar=algo.calendars.US_EQUITIES
    )
```

PY

QuantConnect

```
class MyAlgorithm(QCAlgorithm):
    def my_scheduled_function(self):
        pass

    # Runs a function every day 60 minutes before SPY ETF's market close
    def Initialize(self) -> None:
        self.Schedule.On(
            self.DateRules.EveryDay(),
            self.TimeRules.BeforeMarketClose("SPY", 60),
            self.my_scheduled_function
       )
```

PY

You can read more about scheduled events at the Scheduled Events documentation page.

Setting Slippage and Commissions

QuantConnect supports applying slippage and commissions per individual security.

To add a slippage model to a security, we must set the `SlippageModel` property of the `Security` object to our model. A `Security` is returned when data is added to the algorithm. Examples of setting slippage in an algorithm is provided below for both Quantopian and QuantConnect.

Quantopian

```
import quantopian.algorithm as algorithm
from zipline.finance.slippage import FixedSlippage


    def initialize(context):
        # This will set slippage for all U.S. equities
        algorithm.set_slippage(us_equities=FixedSlippage(0.01))
```

PY

QuantConnect

```
class MyAlgorithm(QCAlgorithm)
    def Initialize(self) -> None:
        spy = self.AddEquity("SPY", Resolution.Minute)
        # `ConstantSlippageModel` is the same as `FixedSlippage` on Quantopian.
        # This will set slippage only for the "SPY" security.
        spy.SlippageModel = ConstantSlippageModel(0.01)
```

PY

Commissions

To add a fee model (otherwise known as Commissions on Quantopian) to a security, we must set the `FeeModel` property of the `Security` object to our model. Examples are provided below for Quantopian and QuantConnect.

Quantopian

```
import quantopian.algorithm as algorithm
from zipline.finance.commission import PerShare


    def initialize(context):
        # Approximates Interactive Brokers' equities trading commissions.
        algorithm.set_commission(us_equities=Pershare(cost=0.0075))
```

QuantConnect

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
       aapl = self.AddEquity("AAPL", Resolution.Minute)
       spy = self.AddEquity("SPY", Resolution.Minute)

       # Sets the fee model to the IB fee model, which simulates
       # the fees we would encounter in live trading.
       spy.FeeModel = InteractiveBrokersFeeModel()

       # Sets the fee model to a constant fee model
       aapl.FeeModel = ConstantFeeModel(0.01, "USD")
```

You can read more about Securities and their attributes at the Securities and Portfolio and Reality Modeling documentation pages.

Manual Asset Lookup

On QuantConnect, we provide the `Symbol` class to reference a security across time with a changing ticker. This is similar to Quantopian's `symbol()` function, which enables the same functionality.

When you add data to your algorithm, a `Security` object is returned. The `Symbol` is accessible from the `Security` as shown below.

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2000, 1, 1)
        self.SetEndDate(2020, 1, 1)
        # TWX will change to AOL when backtesting in the early 2000s, and will
        # change to TWX as the backtest advances past the date it was renamed.
        twx_security = self.AddEquity("TWX", Resolution.Minute)
        twx_symbol = twx_security.Symbol
```

To manually create a reference to an old equity, you can use the `Symbol.Create()` method, or the `Symbol.CreateEquity()` method.

You can read more about Symbols and Security Identifiers (SID) at the Security Identifiers documentation page.

Zipline

Using Data

Pipeline in Algorithms

As covered in the Initialization section, Pipelines can be replicated using Universe Selection in QuantConnect, albeit with some additional steps in between and a performance impact.

In this section, we will construct and define an equivalent pipeline model using universe selection in QuantConnect. We will filter our data set in Coarse and Fine, and apply an additional filter to historical data with a rolling window.

We first create a skeleton algorithm definition to begin setting up our Pipeline. Note that this algorithm will allow all equities through, which will have a substantial performance impact on our algorithm.

```py
class MyPipelineAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2020, 1, 1)
        self.SetEndDate(2020, 10, 20)

        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)

    def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        # Allows all Symbols through, no filtering applied
        return [coarse_data.Symbol for coarse_data in coarse]

    def FineSelectionFunction(self, fine: List[FineFundamental]) -> List[Symbol]:
        # Allows all Symbols through, no filtering applied
        return [fine_data.Symbol for fine_data in fine]
```

The skeleton algorithm is the equivalent of the Pipeline call below.

```py
from quantopian.pipeline import Pipeline
from quantopian.pipeline.domain import US_EQUITIES
from quantopian.research import run_pipeline

pipe = Pipeline(columns={}, domain=US_EQUITIES)
run_pipeline(pipe, '2020-01-01', '2020-10-20')
```

The equivalent of `Pipeline(screen=...)` resolves to the filter applied at the Coarse and Fine stages of universe selection. Let's define a filter of stocks with a dollar volume greater than $50000000 USD, as well as a rolling thirty day return greater than 2%. Once we've initially filtered the Symbols in Coarse Universe Selection, let's define a final filter only allowing stocks with EPS greater than 0. Beware of making `History()` calls with many Symbols. It could potentially cause your algorithm to run out of system resources (i.e. RAM) and reduce performance of your algorithm on universe selection.

```py
from datetime import datetime, timedelta

class MyPipelineAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2020, 1, 1)
        self.SetEndDate(2020, 10, 20)

        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)

    def CoarseSelectionFunction(self, coarse: List[CoarseFundamental]) -> List[Symbol]:
        # Allows all Symbols through, no filtering applied
        dollar_volume_filter_symbols = [coarse_data.Symbol for coarse_data in coarse if
coarse_data.DollarVolume > 50000000]

        # Make a history call to calculate the 30 day rolling returns
        df = self.History(dollar_volume_filter_symbols, self.Time - timedelta(days=60), self.Time,
Resolution.Daily)

        # Compute the rolling 30 day returns
        df = df['close'].groupby(level=0).filter(lambda x: len(x) >= 30).groupby(level=0).apply(lambda x:
(x.iloc[-1] - x.iloc[-30]) / x.iloc[-30])

        # Finally, apply our filter
        dataframe_screen = df[df > 0.02]

        # Filters out any Symbol that is not in the DataFrame
        return [s for s in dollar_volume_filter_symbols if str(s) in dataframe_screen]

    def FineSelectionFunction(self, fine: List[FineFundamental]) -> List[Symbol]:
        # We receive the filtered symbols from before, and we filter by EPS > 0 in this step
       return [s.Symbol for s in fine if s.EarningReports.BasicEPS.ThreeMonths > 0]
```

This class definition is now roughly equivalent to the following `CustomFactor` and Pipeline call in Quantopian, excluding the EPS filtering.

```py
from quantopian.pipeline.filters import QTradableStocksUS
from quantopian.pipeline.factors import AverageDollarVolume

class PercentageChange(CustomFactor):
    def compute(self, today, assets, out, values):
        out[:] = (values[-1] - values[0]) / values[0]

    dollar_volume = AverageDollarVolume(window_length=5)
    dollar_volume_filter = (dollar_volume > 50000000)

    pipe = Pipeline(
        columns={
            "percent_change": PercentageChange(inputs=[USEquityPricing.close], window_length=30)
        },
        screen=(QTradableStocksUS() & dollar_volume_filter)
    )
```

An example of the shape of the DataFrame returned from History is shown below. The DataFrame has a MultiIndex, with `level=0` being the Symbol, and `level=1` being the Time for that point of data. You can index the Symbol/ `level=0` index by using either the SecurityIdentifier string (e.g. `df.loc["AAPL R735QTJ8XC9X"]`) or with the ticker of the Symbol (e.g. `df.loc["AAPL"]`)

```
                           close    high     low    open      volume
symbol time
IBM    2019-04-26 00:00:00  138.62  139.82  137.70  139.82   1419038.0
       2019-04-27 00:00:00  139.44  139.89  138.81  139.28   1275702.0
AAPL   2019-04-26 00:00:00  205.24  207.76  205.12  206.84  10105760.0
       2019-04-27 00:00:00  204.29  205.00  202.12  204.83   9984854.0
```

BarData Lookup

Similar but different, the Quantopian `BarData` object, and the QuantConnect `Slice` object both provide data to the user's algorithm as point-in-time data.

In Quantopian, data is handled via the `handle_data(context, data)` function. In QuantConnect, data is handled via the `OnData(self, slice)` method. Both of these functions accept data whenever new data exists for a given point in time. Although these two functions share the same method signature, the handling of the data is different.

`BarData` vs. `Slice`

`BarData` is the primary mechanism to retrieve the point-in-time data, as well as requesting history for any given securities in Quantopian. The following code retrieves daily historical data from 30 days into the past, as well as getting the most recent data for AAPL at the current point-in-time.

Quantopian

```
def initialize(context):
    context.aapl = sid(24)

def handle_data(context, data):
    # Gets a DataFrame of AAPL history going back 30 days
    aapl_history = data.history(context.aapl, fields=["open", "high", "low", "close", "volume"], 30, "1d")
    # Gets a pandas Series of the most recent AAPL data
    aapl_current = data.current(context.aapl, fields=["open", "high", "low", "close", "volume"])
```

QuantConnect

```
from datetime import timedelta

class MyHistoryAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.aapl = self.AddEquity("AAPL", Resolution.Daily)

    def OnData(self, slice: Slice) -> None:
        # Gets a DataFrame of AAPL history going back 30 days
        aapl_history = self.History([self.aapl.Symbol], timedelta(days=30), Resolution.Daily)

        # Gets the most recent AAPL Trade data (OHLCV)
        aapl_current = slice.Bars[self.aapl.Symbol]
```

`Slice` represents a single point in time, and does not provide the functionality to access historical data itself. To access historical data in an algorithm, use the algorithm's `self.History()` call to request a pandas DataFrame of historical data. In `Slice` , the data that is accessed is not a pandas DataFrame, but rather a single object containing data for a single point in time.

The `TradeBar` class, for example, only contains scalar values of OHLCV, rather than return a DataFrame of OHLCV values. Since the data `Slice` contains is point-in-time, there will be only a single trade/quote bar per Symbol whenever `OnData(self, data)` is called.

QuantConnect provides Quote (NBBO) data for use in your algorithm, otherwise known as a `QuoteBar` . Quote data is only accessible when an algorithm is set to consume Tick, Second, or Minutely data.

You can access Trade (OHLCV) data by accessing the `Bars` attribute of `Slice` . You can access Quote (Bid(OHLCV), Ask(OHLCV)) data by accessing the `QuoteBars` attribute of `Slice` .

Both of the `Bars` and `QuoteBars` attributes are similar to Python dictionaries, and can be used as such. To check to see if there exists a new piece of data for a given security, you can use Python's `in` operator on `Bars` and or `QuoteBars` . You can also choose to iterate on all data received by calling the `Values` attribute of the `Bars` or `QuoteBars` attributes, which will return either a list of `TradeBar` or `QuoteBar` objects.

The `TradeBar` object contains the `Open` , `High` , `Low` , `Close` , `Volume` , `Time` , `EndTime` , and `Symbol` attributes. The `QuoteBar` object contains the following attributes:

- `Bid.Open` , `Bid.High` , `Bid.Low` , `Bid.Close` , `LastBidSize`
- `Ask.Open` , `Ask.High` , `Ask.Low` , `Ask.Close` , `LastAskSize`
- `Time` , `EndTime` , and `Symbol` .

Note that the `Bid` and `Ask` attributes can potentially be `None` if no bid/ask data exists at a given point-in-time.

The example below shows the different ways to access `TradeBar` and `QuoteBar` data, as well as requesting 30 days of AAPL historical data.

PY

```
from datetime import datetime, timedelta


class MyDataAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.aapl_security = self.AddEquity("AAPL", Resolution.Daily)
        self.aapl_symbol = self.aapl_security.Symbol

    def OnData(self, slice: Slice) -> None:
        # Gets 30 days of AAPL history
        aapl_history = self.History([self.aapl_symbol], timedelta(days=30), Resolution.Daily)

        # We must first check to make sure we have AAPL data, since this point-in-time
        # might not have had any trades for AAPL (this is in case you trade a low
        # liquidity asset. The data can potentially be missing for a point-in-time).
        if self.aapl_symbol in slice.Bars:
            aapl_current_trade = slice.Bars[self.aapl_symbol]
            Log(f"{self.Time} :: TRADE :: {self.aapl_symbol} - O: {aapl_current_trade.Open} H:
{aapl_current_trade.High} L: {aapl_current_trade.Low} C: {aapl_current_trade.Close} V:
{aapl_current_trade.Volume}")

        # Check to make sure we have AAPL data first, since there might not have
        # been any quote updates since the previous for AAPL (this is in case you trade
        # a low liquidity asset. The data can potentially be missing for a point-in-time).
        if self.aapl_symbol in data.QuoteBars:
            aapl_current_quote = slice.QuoteBars[self.aapl_symbol]
            if aapl_current_quote.Bid is not None:
                Log(f"{} :: QUOTE :: {} - Bid O: {} Bid H: {} Bid L: {} Bid C: {} Bid size: {}".format(
                    str(self.Time),
                    str(self.aapl_symbol)
                    str(aapl_current_quote.Bid.Open),
                    str(aapl_current_quote.Bid.High),
                    str(aapl_current_quote.Bid.Low),
                    str(aapl_current_quote.Bid.Close),
                    str(aapl_current_quote.LastBidSize)
                ))

            if aapl_current_quote.Ask is not None:
                Log(f"{} :: QUOTE :: {} - Ask O: {} Ask H: {} Ask L: {} Ask C: {} Ask size: {}".format(
                    str(self.Time),
                    str(self.aapl_symbol),
                    str(aapl_current_quote.Ask.Open),
                    str(aapl_current_quote.Ask.High),
                    str(aapl_current_quote.Ask.Low),
                    str(aapl_current_quote.Ask.Close),
                    str(aapl_current_quote.LastAskSize)
                ))
```

Zipline

Ordering

Placing Orders

Both Quantopian and QuantConnect offer several methods for placing orders. Both platforms have `MarketOrder` , `LimitOrder` , `StopOrder` , and `StopLimitOrder` classes to create specific order types, although the `StopOrder` order is named `StopMarketOrder` on our platform. On Quantopian, creating these orders is done by passing a `style` argument to the `order` method.

```py
from zipline.finance.execution import (
    LimitOrder,
    MarketOrder,
    StopLimitOrder,
    StopOrder,
)

def initialize(context):
    context.stock = sid(8554)
    context.ordered = False

def handle_data(context, data):
    if not context.ordered:
        close = data.current(context.stock, 'close')
        order(context.stock, 10, style=MarketOrder())
        order(context.stock, 10, style=LimitOrder(limit_price=close * 0.9))
        order(context.stock, -10, style=StopOrder(stop_price=close * 0.85))
        order(context.stock, -10, style=StopLimitOrder(limit_price=close * 0.75, stop_price=close * 0.85))
        context.ordered = True
```

On QuantConnect, the same orders can be created with

```py
class MyAlgorithm(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2020, 3, 1)
        self.SetCash(100000)
        self.symbol = self.AddEquity("SPY", Resolution.Minute).Symbol
        self.ordered = False

    def OnData(self, slice: Slice) -> None:
        if not self.ordered and slice.ContainsKey(self.symbol):
            close = slice[self.symbol].Close
            self.MarketOrder(self.symbol, 10)
            self.LimitOrder(self.symbol, 10, close * 0.9)
            self.StopMarketOrder(self.symbol, -10, close * 0.85)
            self.StopLimitOrder(self.symbol, -10, close * 0.85, close * 0.75)
            self.ordered = True
```

In addition to the order types above, QuantConnect has several other order types available. Refer to our Trading and Orders documentation for a comprehensive list.

Quantopian's `order_optimal_portfolio` method computes the optimal portfolio weights using an objective and constraints, then places the orders to achieve the desired portfolio. For example, the code below uses the Quantopian

API to create an equal-weighted dollar-neutral portfolio.

```
import quantopian.algorithm as algo
import quantopian.optimize as opt

objective = opt.TargetWeights(weights)
constraints = [
    opt.MaxGrossExposure(1.0),
    opt.DollarNeutral()
]
algo.order_optimal_portfolio(objective, constraints)
```

To acheive the same functionality with QuantConnect, we utilize portfolio construction models . For instance, to create an equal-weighted dollar-neutral portfolio, we could define the following portfolio construction model and attach it to the algorithm.

```
class MyAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.SetStartDate(2020, 5, 1)
        self.SetCash(100000)

        self.SetPortfolioConstruction(EqualWeightedDollarNeutralPortfolioConstructionModel())


class EqualWeightedDollarNeutralPortfolioConstructionModel(PortfolioConstructionModel):

    def DetermineTargetPercent(self, activeInsights: List[Insight]) -> Dict[Insight, float]:
        result = {}

        longs = 0
        shorts = 0
        for insight in activeInsights:
            if insight.Direction == InsightDirection.Up:
                longs += 1
            elif insight.Direction == InsightDirection.Down:
                shorts += 1
            result[insight] = 0

        if longs == 0 or shorts == 0:
            return result

        for insight in activeInsights:
            if insight.Direction == InsightDirection.Up:
                result[insight] = 0.5 / longs
            elif insight.Direction == InsightDirection.Down:
                result[insight] = -0.5 / shorts
        return result
```

When algorithms require manual control of their orders, both Quantopian and QuantConnect have lower-level ordering methods. Quantopian's `order` method, which places an order for a fixed number of shares, directly maps to our `Order` method. Additionally, Quantopian's `order_target_percent` method places an order to adjust a position to a target percent of the current portfolio value. Here's an example use case of these methods on the Quantopian platform.

```
order_target_percent(sid(8554), 10) # Allocate 10% of portfolio
order(sid(8554), 10) # Order 10 shares
```

On QuantConnect, these same orders can be placed using the `SetHoldings` and `Order` methods.

```
self.SetHoldings(symbol, 0.1) # Allocate 10% of portfolio
self.Order(symbol, 10) # Order 10 shares
```

Quantopian and QuantConnect don't have equivalents for all the ordering techniques, although we can create a workaround for most situations. For instance, Quantopian's `order_percent` method places an order in the specified asset corresponding to the given percent of the current portfolio value.

```
order_percent(sid(8554), 10) # Allocate 10% more of portfolio
```

To accomplish this on QuantConnect, we can determine the weight of the asset in the `Portfolio` , then manually determine the new target percent to pass to `SetHoldings` .

```
current_weight = self.Portfolio[symbol].HoldingsValue / self.Portfolio.TotalPortfolioValue
self.SetHoldings(symbol, current_weight + 0.1) # Allocate 10% more of portfolio
```

The Porfolio Object

Quantopian's `Portfolio` class provides read-only access to the current portfolio state. The state includes starting cash, current cash, portfolio value, and positions. The `Portfolio` object is accessed through the `context` object, as seen below.

```
context.portfolio.starting_cash     # Amount of cash in the portfolio at the start of the backtest
                 .cash              # Amount of cash currently held in portfolio
                 .portfolio_value   # Current liquidation value of the portfolio's holdings
                 .positions         # Dict-like object for currently-held positions
```

All of this information, and more, is attainable on QuantConnect by using our `Portfolio` object. Listed below is an example of doing so. Although there isn't a property for starting cash, this value can be saved during Initialization to be referenced later.

```
self.Portfolio.Cash                 # Sum of all currencies in account (only settled cash)
              .TotalPortfolioValue  # Portfolio equity
              .Keys                 # Collection of Symbol objects in the portfolio
              .Values               # Collection of SecurityHolding objects in the portfolio
```

On Quantopian, by iterating through the `positions` dictionary, we can access information about currently-held positions. This includes the number of shares held, the last traded price & date of the asset, and the position's cost basis.

```
for sid, position in context.portfolio.positions.iteritems():
    position.amount            # Number of shares in the position
            .cost_basis        # Volume weighted average price paid per share
            .last_sale_price   # Price at last sale of the asset on the exchange
            .last_sale_date    # Date of last sale of the asset on the exchange
```

Since the `Portfolio` on QuantConnect is a `Dictionary<Symbol, SecurityHolding>` , we can get information on current positions by indexing the `Portfolio` object with a `Symbol` or ticker. The `SecurityHolding` object that is returned contains information related to a single security in the portfolio. For instance, we have

```
self.Portfolio[symbol].Quantity                # Number of shares held
                       .AveragePrice           # Average price of shares held (aka cost basis)
                       .Price                  # Last traded price of the security
```

For more information on our `Portfolio` object refer to the Securities and Portfolio documentation.

Zipline

Logging and Plotting

Logging

In QuantConnect, the `self.Debug()` method is the equivalent of the `log.info()` function in Quantopian. Debug output is viewable via the QuantConnect Terminal Console.

Output to the console via `self.Debug()` is rate-limited. If you need long-term storage of your logs and don't want to overflow the console, consider using `self.Log()` instead. An example comparing logging between Quantopian and QuantConnect is shown below.

Quantopian

```py
def initialize(context):
    log.info("Hello, console!")
```

QuantConnect

```py
class MyLoggingAlgorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.Debug("Hello, console!")
        self.Log("Hello, backtesting logs!")
```

Logging via `self.Log()` has daily limits in QuantConnect. On the free tier, users are limited to 10KB of logs per day for your account/organization. To increase this limit, you can upgrade your account to a different tier. Calls to `self.Debug()` will also count towards the 10KB log limit, but are always outputted to the Terminal Console regardless of your daily log limit.

To view pricing and upgrade your account, visit the Organization Pricing page.

You can view the logs generated via `self.Log()` for a backtest by first accessing a backtest, and clicking the "Logs" tab near the bottom of the page. To download the logs, you can click the "Download Logs" link to download a text file containing your backtest's logs.

Plotting

In Quantopian, plotting/charting is done with the `record(series1_name=scalar_value, series2_name=scalar_value, ...)` function. In QuantConnect, plotting is accessible via the `self.Plot("chart name", "series name", scalar_value)` method.

Plotting multiple series in one function call is possible in Quantopian. However, in QuantConnect, a separate call is needed for each series requiring an update. All series with the same chart name will appear on the same window/pane. New charts can be created by calling `self.Plot()` method with a unique chart name.

An example of plotting in Quantopian vs. QuantConnect is shown below. The QuantConnect code will create two charts, one named "Chart1" with two series of data, and another named "Chart2" with one series of data. Note that you can also use `self.Plot("series name", value)` to place the custom series on the same chart as the equity curve.

Quantopian

```py
import numpy as np
from quantopian.algorithm import record

    def initialize(context):
        context.i = 0

    def handle_data(context, data):
        context.i += 0.25

        record(
            Series1=np.sin(context.i),
            Series2=np.cos(context.i),
            Series3=np.tan(context.i)
        )
```

QuantConnect

```py
import numpy as np

class Algorithm(QCAlgorithm):
    def Initialize(self) -> None:
        self.i = 0

    def OnData(self, slice: Slice) -> None:
        self.i += 0.25

        self.Plot('Chart1', 'Series1', np.sin(self.i))
        self.Plot('Chart1', 'Series2', np.cos(self.i))
        self.Plot('Chart2', 'Chart2 Series1', np.tan(self.i))
```

For more information on plotting in QuantConnect, visit the Charting documentation page.

Zipline

Quick Reference

User-Implemented Functions

The following table describes functions that are treated specially when defined in the IDE.

Algorithms are required to implement one method: Initialize(self).

An optional scheduled event can be defined to behave like before_trading_start(). OnData() is also optional but is treated especially if defined.

Quantopian	**QuantConnect**
`initialize` (context)	`Initialize` (self)
Required function called once at the start of a backtest.	
`before_trading_start` (context, data)	`self.Schedule.On(self.DateRules.EveryDay(), self.TimeRules.AfterMarketOpen(symbol, 10), self.before_trading_start)`
Optional function called prior to market open on every trading day.	
`handle_data` (context, data)	`OnData` (self, slice)
Optional function called at the end of each market minute.	

Interface Classes

The following table describes classes that are passed to `initialize()` , `handle_data()` , `before_trading_start()` and any scheduled with `schedule_function()` .

In QuantConnect/Lean, the portfolio state and the positions are class attributes of QCAlgorithm, available via self keyword. The scheduled function is parameterless and can algorithms can self.CurrentSlice.

Quantopian	QuantConnect
`quantopian.algorithm.interface.AlgorithmContext`	`QCAlgorithm`
Shared object for storing custom state.	
`quantopian.algorithm.interface.BarData`	`Slice`
Provides methods for accessing minutely and daily price/volume data from Algorithm API functions.	
`quantopian.algorithm.interface.Portfolio`	`self.Portfolio`
Object providing read-only access to current portfolio state.	
`quantopian.algorithm.interface.Positions`	`self.Transactions`
A dict-like object containing the algorithm's current positions.	

The features from the BarData methods are found in the Slice object (current data), History method (past data) and class attribute Securities.

Quantopian	QuantConnect
`quantopian.algorithm.interface.BarData.current`	`Slice[symbol]`
Returns the "current" value of the given fields for the given assets at the current simulation time.	
`quantopian.algorithm.interface.BarData.history`	`self.History(symbols, bar_count)`
Returns a trailing window of length `bar_count` containing data for the given assets, fields, and frequency.	
`quantopian.algorithm.interface.BarData.can_trade`	`self.Securities[symbol].IsTradable`
For the given asset or iterable of assets, returns True if all of the following are true:	
`quantopian.algorithm.interface.BarData.is_stale`	`not Slice[symbol].IsFillForward`
For the given asset or iterable of assets, returns True if the asset is alive and there is no trade data for the current simulation time.	

Pipeline

The following table describes functions used to schedule and retrieve results of Universe selection. For more information on the Universe Selection API, see the Univeres API Reference.

Quantopian	QuantConnect
`quantopian.algorithm.attach_pipeline` (...)	`universe = self.AddUniverse` (...) name = universe.Configuration.Symbol
Register a pipeline to be computed at the start of each day.	
`quantopian.algorithm.pipeline_output` (name)	`self.UniverseManager` [name]
Get results of the pipeline attached by with `name` .	

Scheduling Functions

The following tables describes functions that can be used to schedule functions to run periodically during your algorithm.

Quantopian	QuantConnect
`quantopian.algorithm.schedule_function` (func)	`self.Schedule.On(IDateRule, ITimeRule, func)`
Schedule a function to be called repeatedly in the future.	

Date Rules

The following table contains functions that can be used with the `date_rule` parameter of `schedule_function()` .

Quantopian	QuantConnect
`quantopian.algorithm.date_rules.every_day` ()	`self.DateRules.EveryDay(symbol)`
Create a rule that triggers every day.	
`quantopian.algorithm.date_rules.month_start` ([...])	`self.DateRules.MonthStart(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days after the start of each month.	
`quantopian.algorithm.date_rules.month_end` ([...])	`self.DateRules.MonthEnd(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days before the end of each month.	
`quantopian.algorithm.date_rules.week_start` ([...])	`self.DateRules.WeekStart(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days after the start of each week.	
`quantopian.algorithm.date_rules.week_end` ([...])	`self.DateRules.WeekEnd(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days before the end of each week.	

Time Rules

The following table contains functions that can be used with the `time_rule` parameter of `schedule_function()` .

Quantopian	QuantConnect
`quantopian.algorithm.time_rules.market_open` ([...])	`self.TimeRules.AfterMarketOpen(symbol, minutes_after_open = 0)`
Create a rule that triggers at a fixed offset from market open.	
`quantopian.algorithm.time_rules.market_close` ([...])	`self.TimeRules.BeforeMarketClose(symbol, minutes_before_close = 0)`
Create a rule that triggers at a fixed offset from market close.	

Ordering

There are several ways to place orders from an algorithm. For most use-cases, we recommend using SetHoldings method which correctly accounts the porfolio buying power model and fees. At the moment, there is no equivalent for `order_optimal_portfolio()` . However a similar behavior can be replicated with a Portfolio Construction Model.

Algorithms that require explicit manual control of their orders can use the lower-level ordering functions.

Placing Orders

Quantopian	QuantConnect
`quantopian.algorithm.order_optimal_portfolio` (...)	`N/A`
Calculate an optimal portfolio and place orders toward that portfolio.	
`quantopian.algorithm.order` (asset, amount[, ...])	`self.Order(symbol, amount)`
Place an order for a fixed number of shares.	
`quantopian.algorithm.order_value` (asset, value)	`N/A`
Place an order for a fixed amount of money.	
`quantopian.algorithm.order_percent` (asset, ...)	`self.SetHoldings(symbol, percent)`
Place an order in the specified asset corresponding to the given percent of the current portfolio value.	
`quantopian.algorithm.order_target` (asset, target)	`N/A`
Place an order to adjust a position to a target number of shares.	
`quantopian.algorithm.order_target_value` (...)	`N/A`
Place an order to adjust a position to a target value.	
`quantopian.algorithm.order_target_percent` (...)	`self.SetHoldings(symbol, percent)`
Place an order to adjust a position to a target percent of the current portfolio value.	

Controlling Order Execution

Quantopian	QuantConnect
`zipline.finance.execution.ExecutionStyle`	`FillModel`
Base class for order execution styles.	
`zipline.finance.execution.MarketOrder` ([exchange])	`self.MarketOrder` (symbol, quantity)
Execution style for orders to be filled at current market price.	
`zipline.finance.execution.LimitOrder` (limit_price)	`self.LimitOrder` (symbol, quantity, limit_price)
Execution style for orders to be filled at a price equal to or better than a specified limit price.	
`zipline.finance.execution.StopOrder` (stop_price)	`self.StopMarketOrder` (symbol, quantity, stop_price)
Execution style representing a market order to be placed if market price reaches a threshold.	
`zipline.finance.execution.StopLimitOrder` (...)	`self.StopLimitOrder` (symbol, quantity, stop_price, limit_price)
Execution style representing a limit order to be placed if market price reaches a threshold.	

Managing Existing Orders

Quantopian	QuantConnect
`quantopian.algorithm.get_order` (order_id)	`self.Transactions.GetOrderById` (order_id)
Lookup an order based on the order Id returned from one of the order functions.	
`quantopian.algorithm.get_open_orders` ([asset])	`self.Transactions.GetOpenOrders` (symbol)
Retrieve all of the current open orders.	
`quantopian.algorithm.cancel_order` (order_param)	`self.Transactions.CancelOpenOrders` (symbol)
Cancel an open order.	

Customizing the Simulation

Slippage

Algorithms can customize the simulation of order fills by calling SetSlippageModel() with a ISlippageModel which is any class that implements GetSlippageApproximation method. Unlike Quantopian, the slippage model is a security attribute, not a global attribute. Different securities can have different slippage models.

The default slippage model on QuantConnect is a constant slippage model with zero slippage percent (no slippage).

Quantopian	QuantConnect
`quantopian.algorithm.set_slippage` ([...])	`self.Securities[symbol].SetSlippageModel(ISlippageModel)`
Set the slippage models for the simulation.	
`zipline.finance.slippage.SlippageModel` ()	`N/A (class must implement GetSlippageApproximation method)`
Abstract base class for slippage models.	
`zipline.finance.slippage.FixedBasisPointsSlippage` ([...])	`ConstantSlippageModel(slippage_percent)`
Model slippage as a fixed percentage difference from historical minutely close price, limiting the size of fills to a fixed percentage of historical minutely volume.	
`zipline.finance.slippage.NoSlippage` ()	`ConstantSlippageModel(0)`
A slippage model where all orders fill immediately and completely at the current close price.	
`zipline.finance.slippage.FixedSlippage` ([spread])	`N/A`
Simple model assuming a fixed-size spread for all assets.	
`zipline.finance.slippage.VolumeShareSlippage` ([...])	`N/A`
Model slippage as a quadratic function of percentage of historical volume.	

Commissions

Algorithms can customize the simulation of order fees by calling SetFeeModel() with a IFeeModel which is any class that implements GetOrderFee method. Unlike Quantopian, the fee model is a security attribute, not a global attribute. Different securities can have different fee models.

The default fee model on QuantConnect depends on the security type and the brokerage. For instance, the default fee model for equity is the Interactive Brokers fee model.

Quantopian	QuantConnect
`quantopian.algorithm.set_commission` ([...])	`self.Securities[symbol].SetFeeModel(IFeeModel)`
Sets the commission models for the simulation.	
`zipline.finance.commission.CommissionModel`	`N/A (class must implement GetOrderFee method)`
Abstract base class for commission models.	
`zipline.finance.commission.PerShare` ([cost, ...])	`N/A`
Calculates a commission for a transaction based on a per share cost with an optional minimum cost per trade.	
`zipline.finance.commission.PerTrade` ([cost])	`N/A`
Calculates a commission for a transaction based on a per trade cost.	
`zipline.finance.commission.PerDollar` ([cost])	`N/A`
Model commissions by applying a fixed cost per dollar transacted.	
`zipline.finance.commission.NoCommission`	`FeeModel()`
Model commissions as free.	

Benchmark

The default benchmark is SPY.

Quantopian	QuantConnect
`quantopian.algorithm.set_benchmark` (benchmark)	`self.SetBenchmark` (benchmark)
Set the benchmark asset.	