Zellic

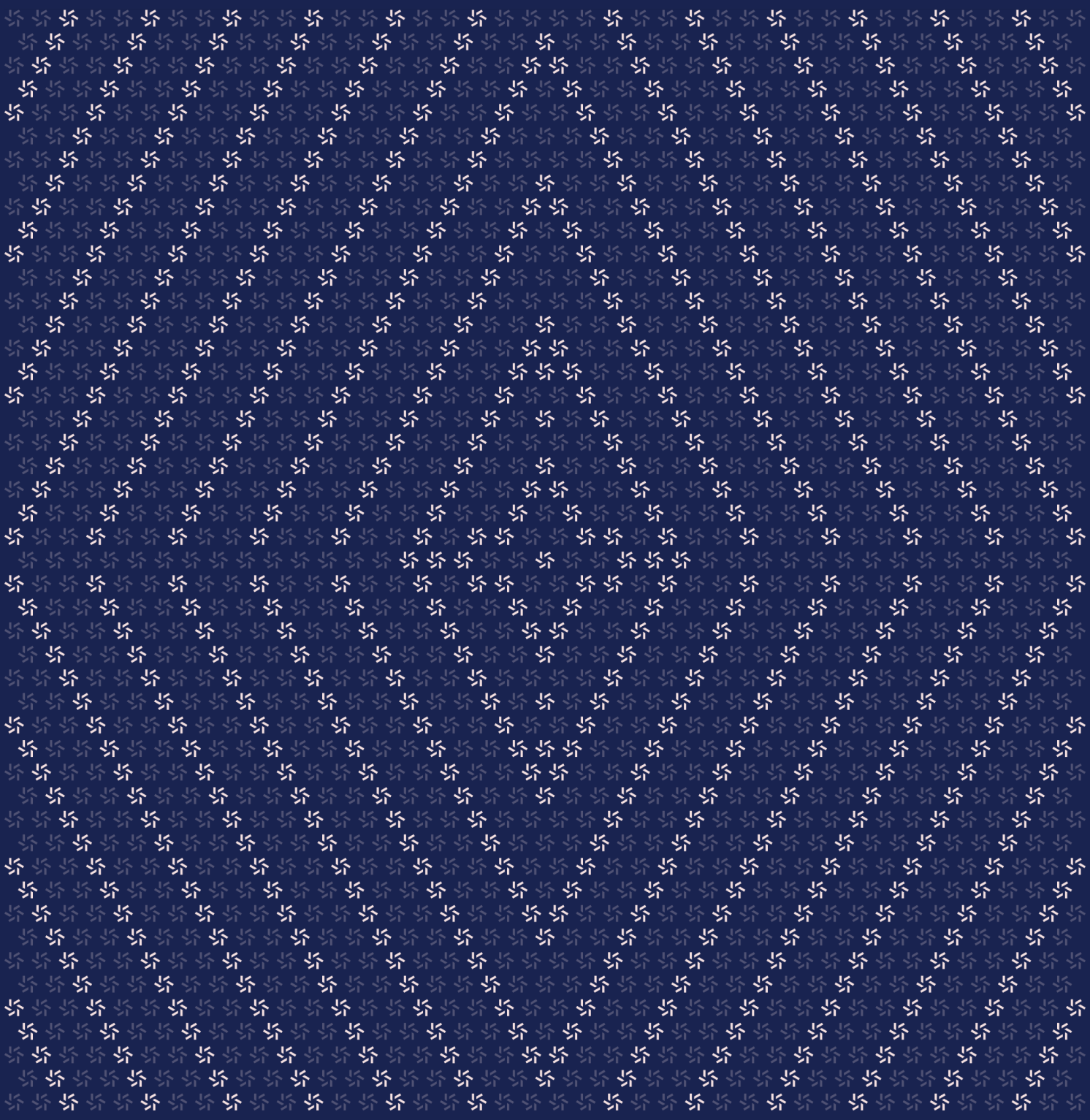
Prepared for
Sam Kazemian
Frax

Prepared by
Kritsada Dechawattana
Jaeeu Kim
Zellic

October 19, 2026

frxUSD Ecosystem

Smart Contract Security Assessment



Contents

About Zellic **6**

1. Overview **7**

1.1. Executive Summary 7

1.2. Goals of the Assessment 7

1.3. Non-goals and Limitations 7

1.4. Results 7

2. Introduction **9**

2.1. About frxUSD Ecosystem 9

2.2. Methodology 9

2.3. Scope 11

2.4. Project Overview 15

2.5. Project Timeline 16

3. Detailed Findings **17**

3.1. Some Tempo fee-paying entry points lack a caller-selected spend cap 17

3.2. OFT transport recipients can decode to zero after source principal is debited 19

3.3. Excess Tempo fees are refunded in LayerZeroEndpointDollar 21

3.4. Unbounded `minters_array` growth can prevent minter removal 23

3.5. A block-scoped oracle cache can outlive feed, scale, and validity changes 25

3.6. Remote-hop sends accept unusable final recipients hidden from OFT validation 27

3.7. UniV2 step prices fee-on-transfer and rebasing inputs at the nominal amount, causing route reverts 29

3.8. Custodian owners can recover active backing while liabilities remain 31

3.9. Initializers accept a zero owner, and `initializeWithPolicy` accepts unusable policies 33

3.10. Rate-limit configuration transitions can reprice historical usage 35

3.11. Custom EIP-712 initializers omit the `onlyInitializing` guard 37

3.12. Internal Hop initializer omits the `onlyInitializing` guard 39

3.13. Freezing the zero address blocks mint and burn operations 41

3.14. Administrator transitions can strand principal awaiting compose settlement 43

3.15. A shared timestamp can discard fractional rate-limit recovery 45

3.16. Executable OFT sends do not enforce the shared-decimal ceiling 47

3.17. Rebinding try-catch catches every revert from `quoteSend()` 49

3.18. Missing fee bounds during initialization 51

3.19. Fee exemption in `max*` functions depends on the caller instead of the queried owner 53

3.20. Missing recipient validation lets swap output be sent to the router or executor 55

4. Discussion 57

4.1. Additional review observations 57

4.2. Limited test coverage 58

5. Threat Model 60

5.1. Module: CustodianWithReceiver.sol 60

5.2. Module: CustodianWithReceiverV2.sol 61

5.3. Module: EIP3009Lib.sol 65

5.4. Module: EIP3009Module.sol 76

5.5. Module: FraxOFTAdapterUpgradeable.sol 97
5.6. Module: FraxOFTMintableAdapterUpgradeable.sol 113
5.7. Module: FraxOFTMintableAdapterUpgradeableTIP20.sol 132
5.8. Module: FraxOFTUpgradeable.sol 157
5.9. Module: FraxOFTUpgradeableTempo.sol 176
5.10. Module: FraxswapRouterMultihopV2.sol 184
5.11. Module: FreezeThawLib.sol 208
5.12. Module: FrxUSD2.sol 221
5.13. Module: FrxUSD3.sol 248
5.14. Module: FrxUSDCustodianBase.sol 251
5.15. Module: FrxUSDCustodianUsdc.sol 258
5.16. Module: FrxUSDCustodianUsdcV2.sol 259
5.17. Module: FrxUSDCustodianWithOracle.sol 263
5.18. Module: FrxUSDOFTUpgradeable.sol 268
5.19. Module: FrxUSDPolicyAdminTempo.sol 302
5.20. Module: HopV201Tempo.sol 309
5.21. Module: LinearRewardsErc4626_2.sol 339
5.22. Module: PancakeV3Plugin.sol 352
5.23. Module: PauseLib.sol 362
5.24. Module: PermitLib.sol 365
5.25. Module: PermitModule.sol 369
5.26. Module: RateLimiterLib.sol 375
5.27. Module: RemoteHopV201Tempo.sol 391
5.28. Module: RemoteMintRedeemHop.sol 405
5.29. Module: RemoteMintRedeemHopTempo.sol 410

5.30. Module: SFrxUSDOFTUpgradeable.sol 415

5.31. Module: SfrxUSD2.sol 431

5.32. Module: SfrxUSD3.sol 449

5.33. Module: TempoAltTokenBase.sol 451

5.34. Module: TempoAltTokenLib.sol 453

5.35. Module: TempoFeeInclusiveWrapper.sol 457

5.36. Module: TempoGasTokenBase.sol 465

5.37. Module: WFRAXTokenOFTUpgradeable.sol 470

5.38. Module: FrxUSDCustodian.sol (in dev-staked-frax-erc4626) 484

5.39. Module: FrxUSDCustodian.sol (in fraxnet-dev) 487

6. Assessment Results 496

6.1. Deployed implementations 496

6.2. Disclaimer 504

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.



1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Frax from September 23rd to October 19th, 2026. During this engagement, Zellic reviewed frxUSD Ecosystem's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could malicious actors steal user funds from the smart contracts?
- Could malicious actors steal funds from the protocol itself?
- Could malfunctioning smart contracts cause loss of funds or other damage to the protocol?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

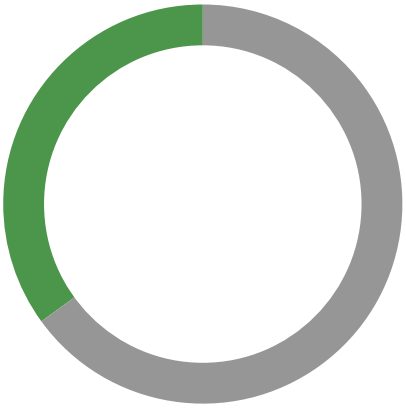
1.4. Results

During our assessment on the scoped frxUSD Ecosystem contracts, we discovered 20 findings. No critical issues were found. Seven findings were of low impact and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Frax in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	7
Informational	13



2. Introduction

2.1. About frxUSD Ecosystem

Frax contributed the following description of frxUSD Ecosystem:

> Frax Finance is a decentralized stablecoin protocol. Its main asset is frxUSD, a stablecoin that can be redeemed for US dollars. frxUSD is backed by USDC and by tokenized real-world assets such as treasury and money-market funds, and these reserves are held by on-chain custodian contracts. sfrxUSD is a savings vault where frxUSD holders deposit their tokens to earn the protocol's yield. Frax tokens can be bridged to more than 20 EVM chains using LayerZero, with Ethereum and Fraxtal holding the locked supply that backs the bridged tokens on other chains. Fraxswap is the protocol's own decentralized exchange.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

frxUSD Ecosystem Contracts

Type	Solidity
Platform	EVM-compatible
Target	frax-tokens
Repository	https://github.com/FraxFinance/frax-tokens ↗
Version	`f0f24fb59fb2ea234afc38e5bb45aeee54c76768`

Programs	`frxUSD/FrxUSD.sol`
	`sfrxUSD/SfrxUSD.sol`
	`sfrxUSD/versioning/SfrxUSD2.sol`
	`sfrxUSD/inherited/LinearRewardsErc4626_2.sol`

Target	frax-tokens (diff from dev-staked-frax-erc4626 @ e7138f9)
Repository	https://github.com/FraxFinance/frax-tokens ↗
Version	`f0f24fb59fb2ea234afc38e5bb45aeee54c76768`

Programs	`ethereum/frxUSD/versioning/FrxUSD3.sol`
	`shared/core/modules/EIP3009Module.sol`
	`shared/core/modules/SignatureModule.sol`
	`ethereum/sfrxUSD/versioning/SfrxUSD3.sol (was StakedFrxUSD3.sol)`

Target	frax-tokens (diff from 6c1c472)

Repository	https://github.com/FraxFinance/frax-tokens ↗
Version	`f0f24fb59fb2ea234afc38e5bb45aeee54c76768`
Programs	`src/contracts/ethereum/frxUSD/versioning/FrxUSD2.sol`
Target	Only changes between b8d76da3a2bb5ff1fc557c194f8fe6496873959d and 7e86f99d18a02262f7d94028ac0f4ba5122ccbd0 (new files in full)
Repository	https://github.com/FraxFinance/frax-oft-upgradeable ↗
Version	`7e86f99d18a02262f7d94028ac0f4ba5122ccbd0`

Programs	`FraxOFTUpgradeable.sol`
	`FraxOFTAdapterUpgradeable.sol`
	`FraxOFTMintableAdapterUpgradeable.sol`
	`WFRAXTokenOFTUpgradeable.sol`
	`frxUsd/FrxUSDOFTUpgradeable.sol`
	`frxUsd/SFrxUSDOFTUpgradeable.sol`
	`FraxOFTUpgradeableTempo.sol`
	`FraxOFTMintableAdapterUpgradeableTIP20.sol`
	`base/TempoAltTokenBase.sol`
	`modules/RateLimiterModule.sol`
	`modules/SupplyTrackingModule.sol`
	`modules/EIP3009Module.sol`
	`modules/FreezeThawModule.sol`
	`modules/PauseModule.sol`
	`modules/PermitModule.sol`
	`modules/signatureModule/EIP712Upgradeable.sol`
	`libraries/RateLimiterLib.sol`
	`libraries/EIP3009Lib.sol`
	`libraries/EIP712Lib.sol`
	`libraries/FreezeThawLib.sol`
	`libraries/PauseLib.sol`
	`libraries/PermitLib.sol`
	`libraries/TempoAltTokenLib.sol`

Target	dev-fraxswap-v3
Repository	https://github.com/FraxFinance/dev-fraxswap-v3 ↗

Version	`cca72784197038d57d31dd0a6b5f23c32c265c95`
Programs	`FraxswapRouterMultihopV2.sol` `PancakeV3Plugin.sol`
Target	hop-v2
Repository	https://github.com/dhruvinparikh/hop-v2 ↗
Version	`02ab5d69c316283590cf63b4fbdea086a79bdd87`

Programs	`hop/RemoteHopV2O1Tempo.sol`
	`base/TempoGasTokenBase.sol`
	`hop/TempoFeeInclusiveWrapper.sol`

Target	hop-v2 (diff from 3dbbbca)
Repository	https://github.com/dhruvinparikh/hop-v2 ↗
Version	`02ab5d69c316283590cf63b4fbdea086a79bdd87`
Programs	`src/contracts/hop/HopV2O1Tempo.sol (was src/contracts/hop/HopV2.sol)`
Target	frax-oft-upgradeable
Repository	https://github.com/FraxFinance/frax-oft-upgradeable ↗
Version	`981a7b96fd85cd0d6e32cd1e1fd46331fbe6aebe`
Programs	`contracts/frxUsd/FrxUSDPolicyAdminTempo.sol`
Target	fraxtal-lz-hop

Repository	https://github.com/dhruvinparikh/fraxtal-lz-hop ↗
Version	31a50544bfd8b80162f035a9b692707f23ed4c65
Programs	hop/RemoteMintRedeemHopTempo.sol base/TempoGasTokenBase.sol
Target	fraxtal-lz-hop (diff from 747051f)
Repository	https://github.com/dhruvinparikh/fraxtal-lz-hop ↗
Version	31a50544bfd8b80162f035a9b692707f23ed4c65
Programs	src/contracts/hop/RemoteMintRedeemHop.sol
Target	dev-staked-frax-erc4626
Repository	https://github.com/FraxFinance/dev-staked-frax-erc4626 ↗
Version	67c54c486a73ae042abcd3b12cff04876d7f0d22
Programs	FrxUSDCustodian.sol FrxUSDCustodianWithOracle.sol
Target	dev-staked-frax-erc4626 (diff from fraxnet-dev @ d24bb1f)
Repository	https://github.com/FraxFinance/dev-staked-frax-erc4626 ↗
Version	67c54c486a73ae042abcd3b12cff04876d7f0d22
Programs	src/contracts/FrxUSDCustodianBase.sol (was src/deps/FrxUSDCustodian.sol)

Target	fraxnet-dev
Repository	https://github.com/FraxFinance/fraxnet-dev ↗
Version	`b1dc549110b1e50e3124f9638905d01a1d749c17`

Programs	`FrxUSDCustodianUsdcV2.sol`
	`CustodianWithReceiverV2.sol`
	`FrxUSDCustodianUsdc.sol`
	`CustodianWithReceiver.sol`

Target	fraxnet-dev (diff from d0e4ef0)
Repository	https://github.com/FraxFinance/fraxnet-dev ↗
Version	`b1dc549110b1e50e3124f9638905d01a1d749c17`
Programs	`src/deps/FrxUSDCustodian.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of five person-weeks. The assessment was conducted by two consultants over the course of three calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Pedro Moura
Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Kritsada Dechawattana
Engineer
kritsada@zellic.io ↗

Jaeeu Kim
Engineer
jaeeu@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 23, 2026	Start of primary review period
September 24, 2026	Kick-off call
October 19, 2026	End of primary review period

3. Detailed Findings

3.1. Some Tempo fee-paying entry points lack a caller-selected spend cap

Target	TempoAltTokenBase-based OFTs, RemoteHopV201Tempo, RemoteMintRedeemHopTempo		
ID	ZELLIC-WWETDM		
Category	Business Logic	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The affected Tempo paths resolve a TIP-20 fee token for the caller and pull it to acquire the endpoint-native `LayerZeroEndpointDollar` fee, but they do not accept a caller-selected maximum TIP-20 spend.

The affected entry points are fixed OFT `send` interfaces, direct `RemoteHopV201Tempo.sendOFT` calls, and the legacy two-argument `RemoteMintRedeemHopTempo.mintRedeem` overload. The fee-inclusive wrapper and four-argument mint/redeem overload already accept bounds.

For a nonwhitelisted fee token, execution obtains an exact-output StablecoinDEX quote, adds bounded settlement headroom, pulls that amount, and refunds the unused balance. Quote and swap are atomic, but execution is not bound to a preview obtained before transaction inclusion. The legacy mint/redeem overload makes this explicit by using `type(uint256).max` as its fee-token cap.

Impact

A public TIP-403–authorized maker can change the executable StablecoinDEX price before transaction inclusion, causing the caller to pay more fee token than previewed if its balance and allowance permit.

In the sampled frxUSD route, moving a peg-priced preview to the -2% tick increases input by about `1 / 0.98 - 1 = 2.0408%` of the endpoint fee. Unused headroom is refunded, insufficient balance or allowance reverts atomically, and bridge principal is unaffected.

Recommendations

Add `feeToken` and `maxFeeTokenAmount` to each affected API and enforce the cap before retaining funds. Entry points that also quote the LayerZero fee internally should accept a caller-selected `maxNativeFee` or prior `MessagingFee`.

Migrate integrations to the bounded mint/redeem overload and remove the legacy overload when compatibility permits. For fixed IOFT compatibility, expose a capped companion entry point. Exact per-call allowances remain useful defense in depth.

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- bf55e95d ↗
- 8f23b072 ↗
- a4880bf1 ↗

3.2. OFT transport recipients can decode to zero after source principal is debited

Target	OFTCoreUpgradeable and Frax OFT implementations		
ID	ZELLIC-4AVT21		
Category	Integration Risks	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The shared OFT source path debits the sender before encoding `SendParam.to`, but it does not validate that the outer transport recipient is usable on the destination:

```
(uint256 amountSentLD, uint256 amountReceivedLD) = _debit(...);
(message, hasCompose) = OFTMsgCodec.encode(_sendParam.to, ...);
```

The destination truncates the wire value to an EVM address and passes it directly to `_credit`:

```
address toAddress = address(uint160(uint256(_message.sendTo())));
uint256 amountReceivedLD = _credit(toAddress, ...);
```

The reviewed minting implementations and adapters pass that address to `_mint` or an underlying-token transfer. If it is zero, destination credit reverts after the source debit has finalized in a separate transaction.

A recipient of `bytes32(0)` is only the canonical trigger. Any nonzero value whose low 160 bits are zero, such as `bytes32(uint256(1) << 160)`, also decodes to `address(0)`. This path is directly reachable through public `OFT.send()` without a hop.

Impact

An initiating sender can burn source OFT supply or lock adapter collateral without completing delivery to a usable destination. The caller selects the recipient and funds the debit; no path was found to alter another sender's recipient or debit a passive holder.

The outcome depends on the destination token. Where credit reaches an OpenZeppelin `_mint`, the zero recipient reverts and the message stalls undelivered. The Ethereum sfrxUSD lockbox is different: its inner token derives from Solmate through SfrxUSD to LinearRewardsErc4626_2, which extends solmate/mixins/ERC4626.sol, and Solmate's `ERC20.transfer` credits the destination without checking it against zero. Delivery to a zero recipient therefore succeeds there,

and the custodied balance becomes permanently unrecoverable at `address(0)` while `totalSupply` is unchanged.

Redirecting zero credit to a sentinel such as `address(0xdead)` would complete the message but make the loss irreversible rather than protect the sender.

Recommendations

Reject `bytes32(0)` in shared OFT quote and send logic. Before debit on an EVM route, also reject a recipient for which `address(uint160(uint256(sendParam.to))) == address(0)`.

Keep this check route-aware because the same `bytes32` field supports non-EVM destinations. A destination-side zero check is useful defense in depth, but the loss-prevention check must run on the source before principal moves.

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- af76ecc1 ↗
- 5fb31cb2 ↗

3.3. Excess Tempo fees are refunded in LayerZeroEndpointDollar

Target	TempoAltTokenBase-based OFTs, RemoteHopV201Tempo, RemoteMintRedeemHopTempo		
ID	ZELLIC-Z3RFKK		
Category	Integration Risks	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

Tempo fee paths accept a caller-selected TIP-20 token, optionally swap it into a whitelisted constituent, and wrap that token into `LayerZeroEndpointDollar` for `EndpointV2Alt`. The endpoint refunds any excess in its configured native ERC-20 wrapper; it does not reverse the conversion into the original caller-selected TIP-20.

```
nativeToken.wrap(targetToken, endpoint, nativeFee);
IOFT(oft).send(sendParam, fee, address(this));
```

Direct OFT callers can choose the refund address, but that address receives `LayerZeroEndpointDollar`. The reviewed hop implementations instead select themselves as the refund recipient.

Impact

A direct caller can receive a different asset than the token debited for fee payment. A contract refund recipient that cannot process or sweep `LayerZeroEndpointDollar` can strand the excess. For hop-originated transfers, the excess accrues to the hop and requires privileged recovery instead of returning automatically to the initiating user.

Exposure is limited to a nonzero fee excess. Same-transaction quote and send logic normally minimizes that amount, the endpoint wrapper is redeemable into a whitelisted constituent, and the bridge principal is unaffected. No third party can independently redirect another caller's refund.

Recommendations

Document the exact refund denomination, and require every selectable refund recipient to support `LayerZeroEndpointDollar`. Attribute hop-held excess to the initiating user, and return it through the normal settlement flow.

If the product promises refunds in the original TIP-20, explicitly unwrap and reverse-swap the excess with a caller-selected minimum output and deadline.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 2e10f596 ↗.

Frax provided the following comment:

> Refunds stay in LZD, a 1:1 claim on a whitelisted stablecoin; reverse-swapping a dust surplus after the message is committed would re-introduce the DEX quote-versus-settlement divergence the fee path pads against. Hop-held refunds are zero by construction: RemoteHopV201Tempo and RemoteMintRedeemHopTempo quote and send in one transaction and the OFT wraps exactly `fee.nativeFee`, so the endpoint's required fee equals the amount supplied and nothing is refunded. There is no caller excess to attribute; swap headroom is already returned to the payer in the fee token in the same call, and `recoverERC20` only covers LZD a third party leaves on the endpoint. For direct callers the refund denomination and recipient requirements will be documented on the public Tempo integration page (Fee Refunds), is documented in the frax-oft-upgradeable README and the NatSpec: pass the freshest quote unmodified, use a refund address that can transfer LZD or call unwrap, and redeem with `unwrap(whitelistedToken, to, amount)`.

3.4. Unbounded `minters_array` growth can prevent minter removal

Target	FrxUSD	ID	ZELLIC-VTXKC7
Category	Code Maturity	Severity	Low
Likelihood	Low	Impact	Low

Description

The `addMinter` function appends every newly authorized minter to `minters_array`. The `removeMinter` function clears authorization in the mapping, linearly scans the complete array, and replaces the matching entry with an `address(0)` tombstone:

```
minters[minter_address] = true;
minters_array.push(minter_address);

for (uint256 i = 0; i < minters_array.length; i++) {
    if (minters_array[i] == minter_address) {
        minters_array[i] = address(0);
        break;
    }
}
```

Removed slots are never reused. Repeated add-and-remove cycles therefore increase the array length and the worst-case scan cost for later entries. The `minters` mapping remains the authoritative authorization state; the array contains zero-address tombstones.

Impact

Only the privileged timelock can grow the array, and the reviewed live length was 10, so there is no immediate gas-limit failure. With sufficient historical growth, however, removing a compromised late-index minter can exceed the block gas limit. The transaction then reverts, including the mapping deletion, preventing revocation.

Recommendations

Use an address-to-index mapping with bounded deletion, or reuse tombstoned slots if stable indexes are required. Also cap and monitor the array length, and retain a mapping-only emergency revocation path whose cost does not depend on historical entries.

Document that `minters` is authoritative and zero array entries are tombstones.

Remediation

This issue has been acknowledged by Frax.

3.5. A block-scoped oracle cache can outlive feed, scale, and validity changes

Target	FrxUSDCustodianWithOracle	ID	ZELLIC-NTTVDX
Category	Business Logic	Severity	Low
Likelihood	Low	Impact	Low

Description

Each state-changing conversion stores one raw oracle answer for the remainder of the block. Every later conversion in that block returns the stored value without checking whether the configured feed, answer, round, decimal scale, or freshness policy changed:

```
if (lastOracleUpdate == block.number) {
    return lastSavedOraclePrice;
}
```

This creates several same-block inconsistencies:

- The `setCustodianOracle` function changes the feed and `oracleDecimals` without invalidating the cached raw answer. The old feed's answer can therefore be interpreted using the new feed's decimal scale.
- An eligible caller can prime the cache with a zero-amount conversion before the existing feed changes its answer. Later conversions in the same block continue to use the old answer.
- Tightening `maximumOracleDelay` or replacing a stale feed does not revalidate an answer already cached in that block.

A searcher can order a zero-amount cache prime before a feed or owner update and execute a favorable conversion after that update in the same block.

Impact

After an otherwise valid feed, answer, or freshness-policy change, a permissionless searcher can convert against stale or incorrectly scaled pricing. A different-scale migration can expose the transferable reserve or remaining mint capacity; same-feed and delay changes can also produce favorable stale conversions.

A fork reproduction primed the cache, installed an economically equivalent feed with a different scale, and withdrew the full sampled reserve with 1 frxUSD. Separate proofs covered same-feed repricing and delay tightening. The required owner-approved migration or same-block feed update makes the trigger rare, but demonstrated loss can be reserve-scale.

Recommendations

Do not persist a raw oracle answer across transactions based only on `block.number`. Read the feed once per deposit, mint, withdraw, or redeem and pass one coherent `(oracle, decimals, answer)` snapshot through that execution.

If caching is retained, bind it to the oracle address, decimal scale, freshness delay, and feed round or update fingerprint. Refresh that tuple atomically on every oracle-policy change, and test same-feed updates, delay tightening, and decimal rotations around a zero-amount cache prime.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 99df4a09 ↗.

3.6. Remote-hop sends accept unusable final recipients hidden from OFT validation

Target	RemoteHopV201Tempo, HopV201Tempo, and TempoFeeInclusiveWrapper		
ID	ZELLIC-CWSWSK		
Category	Integration Risks	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `RemoteHopV201Tempo.sendOFT` function copies the caller-supplied final recipient into `HopMessage` and pulls principal without validating that recipient:

```
HopMessage memory hopMessage = HopMessage({
    srcEid: $.localEid,
    dstEid: _dstEid,
    dstGas: _dstGas,
    sender: bytes32(uint256(uint160(msg.sender))),
    recipient: _recipient,
    data: _data
});

_amountLD = removeDust(_oft, _amountLD);
if (_amountLD > 0) ITIP20(IOFT(_oft).token()).transferFrom(msg.sender,
    address(this), _amountLD);
```

No hop or wrapper entry point rejects `bytes32(0)` or a nonzero value whose low 160 bits decode to `address(0)`. The outcome depends on the route:

- A same-chain local send reaches `_sendLocal` and reverts atomically.
- An empty-data remote route eventually uses the final recipient as an outer OFT `SendParam.to`, exposing the transport-layer failure described separately.
- A data-bearing route addresses the OFT message to a valid hop and keeps the final recipient inside `HopMessage`. Destination credit succeeds, but later `lzCompose` retries all revert when `_sendLocal` decodes the nested recipient to zero.

An outer OFT recipient check cannot detect the composed case because the OFT sees only the valid hop address.

Impact

The initiating caller can commit its principal to an unusable remote intent. On a composed route, principal remains credited to the destination hop while the immutable nested recipient causes every compose retry to fail; recovery or compensation then requires administrator action.

The caller chooses the recipient and funds the transfer. No path was found to alter another user's recipient or debit a passive holder.

Recommendations

Validate the final recipient consistently in hop and wrapper quote/send paths, before any token pull or fee collection. Reject `bytes32(0)` universally, and reject low-160-bit-zero values when the final destination is EVM.

Also validate before `_sendLocal` as defense in depth. Messages already in flight need a claim-directed cancellation or refund path because OFT `_credit` validation cannot inspect a recipient nested in compose data.

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- 8ef37e72 ↗
- 1be0b1ca ↗

3.7. UniV2 step prices fee-on-transfer and rebasing inputs at the nominal amount, causing route reverts

Target	FraxswapRouterMultihopV2	ID	ZELLIC-6249V4
Category	Integration Risks	Severity	Low
Likelihood	Low	Impact	Low

Description

> **Note:** This finding was discovered by Frax during remediation.

The `FraxswapExecutorV2.stepUniV2()` function computes the output from the step's nominal `amountIn`, transfers that amount to the pair, and then calls `swap()` with the precomputed output:

```
amountOut = getAmountOut(amountIn, zeroForOne ? reserve0 : reserve1,
    zeroForOne ? reserve1 : reserve0, feeBps);
if (flags & 16 == 0) TransferHelper.safeTransfer(tokenIn, target, amountIn);
IUniswapV2Pair(target).swap(/* amountOut computed above */);
```

The same applies when the pair quotes through `getAmountOut(amountIn, tokenIn)`, which is also called with the nominal amount.

When `tokenIn` is a fee-on-transfer token, or a rebasing token that delivers 1 to 2 wei less (such as stETH), the pair receives less than `amountIn`. The output was priced for the larger amount, so the pair's invariant check fails and `swap()` reverts.

The route's measure flag does not help. It only changes the swap's basis amount to the executor's actual balance, and the executor-to-pair transfer inside the step is still taxed. In two other cases, there is no executor balance to measure at all:

1. **Direct-funded first swap.** The router pulls the user's tokens straight into the first pair. With the measure flag set, the executor balance is 0 and the step is skipped, so the route reverts with `FSE:IO`.
2. **Pair-to-pair chain.** The previous pair sends its output straight to the next pair through a receiver override. The credited amount is the nominal output, but the next pair receives less.

For rebasing tokens on the default funding path, the executor receives `amountIn - 1` from the router and then tries to forward `amountIn`, which fails with a balance revert. For 1-wei shortfalls, the invariant check only fails when 1 wei of input is worth about 1 wei of output or more.

The contract documentation states that the measure flag is "needed for fee-on-transfer tokens", which `stepUniV2()` cannot deliver.

Impact

Routes whose UniV2 step takes a fee-on-transfer token, or a short-delivering rebasing token, as input always revert, regardless of the measure flag. No funds are lost, because the revert rolls back the whole transaction. These tokens cannot be swapped through UniV2-style steps, contrary to what the documentation implies. Users must use another route, for example a generic call step to a router that supports fee-on-transfer tokens.

Recommendations

Price the input from what the pair actually received. In `stepUniV2()`, send the input first (unless the pair is already funded), and then derive `amountIn` from the pair's `tokenIn` balance above its stored reserves, and quote and call `swap()` with that amount. This also covers direct funding and receiver overrides, because the amount is measured at the pair. For Fraxswap pairs, run `executeVirtualOrders` first and subtract the TWAMM reserves, since the pair holds the input token of pending long-term orders. Consider rejecting this mode on a Fraxswap pair without `executeVirtualOrders`, which would overstate `amountIn` and revert.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 488caac2 ↗.

3.8. Custodian owners can recover active backing while liabilities remain

Target	FrxUSDCustodianBase, FrxUSDCustodian, FrxUSDCustodianWithOracle		
ID	ZELLIC-B8ABDX		
Category	Protocol Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The custodians inherit an owner-only recovery function that can transfer any ERC-20 token, including the active `custodianTkn`, without reducing `frxUSDMinted`:

```
function recoverERC20(address token, uint256 amount) external onlyOwner {
    SafeERC20.safeTransfer(IERC20(token), owner(), amount);
}
```

Normal redemption burns frxUSD and decreases `frxUSDMinted` before transferring backing. Recovery bypasses both steps. Because `maxRedeem` is capped by the remaining token balance, full recovery can leave liability unchanged while reducing redeemable capacity to zero.

A fork proof reproduced that state. A sampled live recovery was surplus-only because local `frxUSDMinted` was zero at the time.

Impact

Recovering active backing can leave local liability without matching reserves. Partial depletion makes holders compete for the remaining balance, and later deposits can fund an existing shortfall.

No non-owner reserve-removal path was identified. This is an explicit trusted-custody and solvency assumption, not an unprivileged exploit.

Recommendations

Separate stray-token rescue from backing withdrawal. If active backing must be immovable, reject `token == address(custodianTkn)`. If rebalancing is required, allow only surplus above a gross, fee-independent liability floor:

```
uint256 required = _convertToAssets(frxUSDMinted, Math.Rounding.Ceil);
uint256 balance = custodianTkn.balanceOf(address(this));
uint256 surplus = balance > required ? balance - required : 0;
if (amount > surplus) revert InsufficientSurplus();
```

Do not use `previewRedeem` for the floor because mutable fees and caller exemptions affect it. Oracle-priced custodians should use one fresh coherent oracle observation. Emit and monitor reserve, liability, and withdrawal values.

Remediation

Frax acknowledged the recovery authority as intended behavior. They stated that its owner multi-signature wallet must be able to manage backing without delay and that frxUSD backing may be held outside the custodians. No code change is planned; unrestricted owner recovery remains an explicit trust and solvency assumption.

3.9. Initializers accept a zero owner, and `initializeWithPolicy` accepts unusable policies

Target	FrxUSDPolicyAdminTempo	ID	ZELLIC-2SEV54
Category	Coding Mistakes	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

Both initialization paths transfer ownership without rejecting `address(0)`. The normal path still creates a BLACKLIST policy administered by the proxy, but no local owner remains to manage freezers, thaw accounts, or transfer policy administration.

The `initializeWithPolicy` function additionally stores an arbitrary policy ID without checking that it exists, uses BLACKLIST semantics, or is administered by the proxy:

```
function initializeWithPolicy(address owner_, uint64 policyId_)
    external initializer {
    _transferOwnership(owner_);
    _getPolicyAdminStorage().policyId = policyId_;
}
```

The `policyExists` modifier rejects only ID zero, while `isFrozen` assumes blacklist semantics. A nonexistent, non-BLACKLIST, or incorrect-admin policy can make freeze/thaw revert or invert `isFrozen`.

Impact

An authorized deployer can initialize either one-shot path with no local owner. The alternate path can also bind an unusable policy, requiring a proxy upgrade or action by the actual policy administrator to recover.

Reviewed production scripts use the normal initializer with a multi-signature owner. The issue therefore requires a privileged deployment mistake, not an unprivileged call against an initialized proxy.

Recommendations

Reject a zero owner in both initializers. For `initializeWithPolicy`, also reject policy ID zero and require an existing BLACKLIST policy whose administrator is `address(this)`.

Initialize atomically, verify that the TIP-20's active policy ID matches the stored ID, and add negative tests for invalid owner, policy type, existence, and administrator.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 5707025d ↗.

Frax provided the following comment:

> Sanity check added for sanity type however the re-deployment is not required because it has a valid self-administered BLACKLIST policy.

3.10. Rate-limit configuration transitions can reprice historical usage

Target	RateLimiterModule, RateLimiterLib		
ID	ZELLIC-PKHXR5		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

Per-endpoint ID (EID) updates checkpoint usage under the old effective configuration before replacement. The default setter only validates and replaces the default, without checkpointing EIDs that inherit it.

On the next state calculation, each inheriting EID applies the new limit and window to all time elapsed since `lastUpdated`, including time that occurred under the old configuration.

For example, an EID that fully consumes a 100-token, 100-second bucket and waits 10 seconds should carry 90 tokens of usage into a replacement configuration. Replacing the default with 1,000 tokens per 10 seconds instead clears the old usage and reports 1,000 available rather than the checkpoint-preserving 910.

A related edge occurs when a disabled per-EID override stores zero limits and windows. Reenabling checkpoints under that zero-valued configuration, old usage is preserved while `lastUpdated` advances, so the disabled interval contributes no refill. Otherwise, inactive placeholder fields therefore determine historical accrual.

Impact

An authorized transition can expose extra capacity, preserve excess usage, or unexpectedly delay refill. A default change affects inheriting EIDs; a zero-valued disable/reenable affects the selected route and direction.

Both cases require owner configuration and grant no capacity beyond the replacement policy. This is a transition-accounting hazard, not an unprivileged rate-limit bypass.

Recommendations

Preserve old-slope accrual without an unbounded scan. One option is a hard-bounded set of default-backed EIDs that is checkpointed before replacement.

Otherwise, retain bounded version history and require each stale EID to apply every intervening

transition before a history slot can be reused.

Define disabled-time semantics explicitly; zero-valued placeholder fields should not decide whether prior usage refills. Test faster and slower replacements, zero-valued disabled overrides, and the intended refill or reset policy.

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- 363d430e ↗
- 2c6ed6dc ↗

3.11. Custom EIP-712 initializers omit the `onlyInitializing` guard

Target	EIP712Upgradeable	ID	ZELLIC-SA8P5Q
Category	Code Maturity	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

The custom upgradable EIP-712 module defines both initialization helpers without OpenZeppelin's `onlyInitializing` modifier:

```
function __EIP712_init(string memory name, string memory version) internal {
    __EIP712_init_unchained(name, version);
}

function __EIP712_init_unchained(string memory name, string memory version)
    internal {
    EIP712Storage storage $ = _getEIP712Storage();
    $._name = name;
    $._version = version;
    $._hashedName = 0;
    $._hashedVersion = 0;
}
```

These helpers rewrite the EIP-712 domain and reset its cached hashes. Standard upgradable modules restrict equivalent writes to an active initializer or reinitializer.

Every reviewed call site, at the time of writing, is inside `reinitializer(3)` or `reinitializer(4)`, and neither helper is ABI-exposed.

Impact

No unprivileged path invokes these helpers outside initialization. The omission instead weakens the inheritance contract: future derived code could expose a call that changes the signature domain and invalidates or changes verification of outstanding permit and EIP-3009 authorizations.

This is upgrade hardening, not a vulnerability in the reviewed call surface.

Recommendations

Add `onlyInitializing` to both helpers:

```
function __EIP712_init(string memory name, string memory version)
    internal
    onlyInitializing
{
    __EIP712_init_unchained(name, version);
}

function __EIP712_init_unchained(string memory name, string memory version)
    internal
    onlyInitializing
{
    // Existing storage writes.
}
```

Retain the guarded external reinitializer functions. Test that initialization succeeds inside them and that a derived external call outside an initialization context reverts.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 22f00aac ↗.

3.12. Internal Hop initializer omits the `onlyInitializing` guard

Target	HopV201Tempo	ID	ZELLIC-TBDDM5
Category	Code Maturity	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

The internal `HopV201Tempo.__init_HopV201()` function runs during initialization, but it has no `onlyInitializing` modifier:

```
function __init_HopV201(
    uint32 _localEid,
    address _endpoint,
    uint32 _numDVNs,
    address _EXECUTOR,
    address _DVN,
    address _TREASURY,
    address[] memory _approvedOfts
) internal {
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);

    HopV2Storage storage $ = _getHopV2Storage();
    $.localEid = _localEid;
    $.endpoint = _endpoint;
    // Additional configuration writes.
}
```

The reviewed call surface is safe as of the time of writing. The only production caller of this `HopV201Tempo` helper is `RemoteHopV201Tempo.initialize()`, which is marked `initializer`. The helper is internal and absent from the external ABI, and the implementation constructors call `_disableInitializers()`. The active protection comes from the current caller and inheritance graph, not from the helper itself.

Impact

A future derived contract could invoke the helper outside initialization, for example from a setter or migration entry point. That path would grant `DEFAULT_ADMIN_ROLE` to its caller, overwrite the hop's scalar routing and fee-infrastructure configuration, and add the supplied OFT approvals.

No such path exists in the reviewed contracts, so this is upgrade and inheritance hardening rather than a vulnerability in the current call surface.

Recommendations

Add `onlyInitializing` to `__init_HopV2O1()`:

```
function __init_HopV2O1(/* existing arguments */)
    internal
    onlyInitializing
{
    // Existing role grant and configuration writes.
}
```

The existing `initializer` entry point establishes the required context, so this change preserves current initialization behavior and adds only a small one-time initialization check. Test that initialization still succeeds and that a derived external call outside initialization reverts with `NotInitializing()`.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 04ccf6d3 ↗.

3.13. Freezing the zero address blocks mint and burn operations

Target	FrxUSD, FrxUSDOFTUpgradeable		
ID	ZELLIC-AYPT3H		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The owner or a delegated freezer can freeze any address, including `address(0)`. Transfer hooks reject an update when either endpoint or the caller is frozen:

```
function freeze(address account) external {
    if (!isFreezer(msg.sender) && msg.sender != owner()) revert NotFreezer();
    _freeze(account);
}

if (isFrozen(to) || isFrozen(from) || isFrozen(msg.sender)) revert IsFrozen();
```

ERC-20 minting uses `from == address(0)`, while burning uses `to == address(0)`. Freezing the zero address therefore blocks every non-owner mint and burn on the affected contract instance. OFT outbound settlement burns and inbound settlement mints, so both bridge directions are also blocked.

Impact

A compromised or mistaken delegated freezer can turn a per-account compliance role into a system-wide issuance, redemption, and bridge-liveness control. Ordinary nonzero-to-nonzero transfers continue, and no assets are transferred to the freezer.

The trigger is privileged and reversible; an available owner can call `thaw(address(0))`, and the owner already has an explicit pause capability. This makes the issue an Informational role-design and liveness concern.

Recommendations

Reject zero in the shared internal freeze primitive so both single and batch entry points inherit the check:

```
if (account == address(0)) revert InvalidFreezeTarget();
```

If delegated freezers are intentionally allowed to halt all mint and burn operations, expose a separately named action, role, and event instead of using the zero address as an implicit global sentinel.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 0eb8908d ↗.

3.14. Administrator transitions can strand principal awaiting compose settlement

Target	RemoteHopV201Tempo and HopV201Tempo		
ID	ZELLIC-CWSWST		
Category	Protocol Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The destination OFT credits a compose recipient before queueing `lzCompose` for a separate transaction. For composed hop transfers, RemoteHopV201Tempo therefore holds principal between destination credit and compose execution:

```
uint256 amountReceivedLD = _credit(toAddress, _toLD(message.amountSD()),
    origin.srcEid);
endpoint.sendCompose(toAddress, guid, 0, composeMsg);
```

Several privileged operations cross this interval:

- The `recoverERC20` function can transfer underlying tokens without reserving balances for queued compose messages.
- The `setMessageProcessed` function can pre-mark the pending tuple; compose then returns as a duplicate and finalizes without recipient payment.
- OFT deapproval or pause makes compose revert until restored.
- Replacing `remoteHop(srcEid)` changes sender interpretation for an already credited message and can make strict recipient authentication revert.

The hop has no pending-GUID reservation or claim ledger tying pooled balances to queued messages.

Impact

Recovery can underfund a pending transfer; replay pre-marking is terminal for automatic retry because compose returns success; and configuration changes can delay settlement until governance restores prior state.

All triggers are privileged. Governance can replenish, restore, or compensate manually, so this is a custody and operational-settlement boundary rather than a permissionless drain.

Recommendations

Track pending claims and reconcile them before recovery, replay pre-marking, OFT deapproval, or remote-hop replacement. Preserve immediate emergency pause and revocation, but define an explicit retry, cancellation, or compensation process.

For on-chain enforcement, reserve amounts by authenticated pending GUID and version trusted-hop configuration for queued messages. Keep a claim-aware emergency recovery path; a blanket token-recovery ban would also block refunds for genuinely failed compose messages.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 79ecfc5a ↗.

3.15. A shared timestamp can discard fractional rate-limit recovery

Target	RateLimiterModule, RateLimiterLib		
ID	ZELLIC-BXHRC7		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

Each EID stores outbound usage, inbound usage, and one shared `lastUpdated` timestamp. Every successful nonzero consumption in either direction projects both usage values and then persists the current timestamp.

```
state.outboundUsage = uint112(_decayUsage(..., elapsed));
state.inboundUsage = uint112(_decayUsage(..., elapsed));
state.lastUpdated = currentTimestamp;
```

The `_decayUsage` function floors each projection independently:

```
uint256 replenished = (limit * elapsed) / window;
```

Opposite-direction traffic can materialize usage before one full local base unit accrues, discard the fractional numerator, and advance the shared timestamp. For example, with a one-unit outbound limit over two seconds, one successful inbound delivery per second can keep saturated outbound usage unchanged across multiple outbound windows.

Impact

Once both directions are enabled, a caller can trigger the effect only by completing real opposite-direction traffic. Each timestamp advancement discards less than one local base unit. With at least six local decimals, even one materialization per second loses less than one micro-token per second, or about 31.6 tokens per year.

Complete stalling requires a dust-scale limit-to-window ratio. The proof of concept used a one-unit limit and a two-second window, and no affected deployed configuration was identified. This is a bounded precision/liveness discrepancy, not a material rate-limit bypass.

Recommendations

Use independent outbound and inbound accrual timestamps, or carry each direction's division remainder across updates. Bind any remainder to the active limit/window version, and clear it on an explicit reset or configuration change.

Measure hot-path storage costs before adopting remainder accounting. Independent timestamps or a minimum configured recovery rate may be simpler.

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 5359fde1 ↗.

3.16. Executable OFT sends do not enforce the shared-decimal ceiling

Target	Frax OFT implementations and adapters		
ID	ZELLIC-BFHKMW		
Category	Integration Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

OFT messages encode transfer amounts in a `uint64` shared-decimal field. The inherited conversion narrows the value without checking representability:

```
function _toSD(uint256 amountLD) internal view returns (uint64 amountSD) {
    return uint64(amountLD / decimalConversionRate);
}
```

The reviewed OFTs and adapters advertise `_toLD(type(uint64).max)` as `quoteOFT().oftLimit.maxAmountLD`, but `_debitView` does not enforce it. A send can burn, lock, or ledger-account the full local-decimal amount before message encoding retains only the low 64 shared-decimal bits. At exactly `2**64` shared units, the payload amount wraps to zero while the source receipt reports the full amount.

Slippage does not close the gap: `_debitView` checks `_minAmountLD` against the full dust-trimmed amount before narrowing. The `quoteSend` function prices the truncated packet, while mintable-adapter accounting records the full source amount.

Impact

An oversized sender can lose its full source amount while the destination receives only the truncated value. The source receipt and transfer ledger retain the full amount, while the fee quote corresponds to the truncated packet.

Reachability is remote. At six shared decimals, the first unrepresentable amount is 18,446,744,073,709.551616 whole tokens. The proof of concept required a public-mint mock, while sampled frxUSD supply was about 89.8 million. Only the initiating sender's balance is debited.

Recommendations

Before narrowing, reject any amount for which `amountReceivedLD / decimalConversionRate > type(uint64).max`. Make `quoteOFT`, `quoteSend`, and `send` agree at the exact boundary.

A checked common `_toSD` path is sufficient for safety because a revert rolls back the earlier debit; an earlier `_debitView` check also avoids wasted gas. Keep the change compact, and verify linked FrxUSDOFTUpgradeable bytecode against EIP-170.

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- 3f4f816b ↗
- 9876b375 ↗

3.17. Rebinding try-catch catches every revert from `quoteSend()`

Target	RemoteMintRedeemHopTempo		
ID	ZELLIC-2AJ02J		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `_quoteSendRebindingOnFailure()` function quotes the OFT through a try-catch. It is meant to recover when LayerZero has delisted the hop contract's own FeeManager fee token (whichever token the previous call paid with), which makes the OFT's quote validation revert.

```
try IOFT(_oft).quoteSend(_sendParam, false) returns (MessagingFee memory _fee)
    {
    return _fee;
} catch {
    _bindFeeToken(_feeToken);
    return IOFT(_oft).quoteSend(_sendParam, false);
}
```

The `catch` block has no condition, so it runs for any revert from `quoteSend()`. That includes failures unrelated to the fee-token binding, such as a missing peer, an unset enforced option, or a misconfigured OFT. For these failures, the function still calls `_bindFeeToken(_feeToken)`. The `_bindFeeToken()` function makes a `userTokens` read and, if the token differs, a `setUserToken` write to the FeeManager precompile.

In the four-argument `mintRedeem()`, `_feeToken` is supplied by the caller. The contract's persistent fee token is therefore temporarily set to a token the caller chose, before it is known whether the binding was the cause. The second `quoteSend()` then fails again for the same unrelated reason, and the whole transaction reverts, which undoes the binding.

Impact

No state change persists and no funds are at risk because a failing call reverts as a whole. These are the costs:

- Every call that fails for an unrelated reason wastes gas on the precompile calls and a second `quoteSend()`.
- The caller sees the error from the second quote, not the first.

- The fallback is broader than its documentation: the NatSpec describes it as handling a delisted binding, but it fires on every failure.

Recommendations

Narrow the fallback to the case it is meant for. Check whether the current binding is still LayerZero-whitelisted before rebinding, instead of reacting to any revert:

```
function _quoteSendRebindingOnFailure(
    address _oft,
    SendParam memory _sendParam,
    address _feeToken
) internal returns (MessagingFee memory fee) {
    address bound = StdPrecompiles.TIP_FEE_MANAGER.userTokens(address(this));
    if (bound == address(0) || !nativeToken.isWhitelistedToken(bound)) {
        _bindFeeToken(_feeToken);
    }
    return IOFT(_oft).quoteSend(_sendParam, false);
}
```

This removes the try-catch, skips the rebind on the success path, and lets unrelated failures revert with their original error. The check `bound == address(0)` matters because an unset binding falls back to pathUSD in the FeeManager, which may not be whitelisted.

If try-catch is preserved, add a comment saying it intentionally covers every quote failure, and note that the binding change is rolled back by the final revert.

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- 018261ce ↗
- 755c85fc ↗

3.18. Missing fee bounds during initialization

Target	FrxUSDCustodian in fraxnet-dev, FrxUSDCustodianBase in dev-staked-frax-erc4626		
ID	ZELLIC-K7354C		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The FrxUSDCustodian and FrxUSDCustodianBase contracts express `mintFee` and `redeemFee` as 18-decimal fractions of the transferred amount. The owner-only `setMintRedeemFee()` enforces that both are strictly below `1e18`:

```
function setMintRedeemFee(uint256 _mintFee, uint _redeemFee)
    public onlyOwner {
  require(_mintFee < 1e18 && _redeemFee < 1e18, "Fee must be a fraction of
    underlying");
  mintFee = _mintFee;
  redeemFee = _redeemFee;
}
```

The `initialize()` function assigns the same variables with no validation:

```
function initialize(address _owner, uint256 _mintCap, uint256 _mintFee,
    uint256 _redeemFee) public {
  ...
  mintFee = _mintFee;
  redeemFee = _redeemFee;
  ...
}
```

All four preview functions compute `1e18 - fee` — for example:

```
function previewMint(uint256 _sharesOut)
    public view virtual returns (uint256 _assetsIn) {
  uint256 fee = mintFee;
  _assetsIn = _convertToAssets(_sharesOut, Math.Rounding.Ceil);
  if (fee > 0) _assetsIn = Math.mulDiv(_assetsIn, 1e18, (1e18 - fee),
```

```
    Math.Rounding.Ceil);
}
```

If a fee is set to exactly `1e18` at initialization, `previewMint()` and `previewWithdraw()` revert with a division-by-zero error. If a fee is greater than `1e18`, computing `1e18 - fee` underflows and reverts in all four preview functions.

Impact

A custodian initialized with an out-of-range fee reverts on every deposit, mint, withdraw, and redeem. No funds are lost or at risk, and an owner who is aware of the problem can recover by calling `setMintRedeemFee()` with valid values. Only the deployer controls the initialization parameters, so an external attacker cannot trigger this.

Recommendations

Apply the same bound in `initialize()` as in `setMintRedeemFee()`:

```
require(_mintFee < 1e18 && _redeemFee < 1e18, "Fee must be a fraction of
    underlying");
```

Remediation

This issue has been acknowledged by Frax, and fixes were implemented in the following commits:

- 05dbeade ↗
- e788e81d ↗

3.19. Fee exemption in `max*` functions depends on the caller instead of the queried owner

Target	FrxUSDCustodianWithReceiverV2 and FrxUSDCustodianUsdcV2 in fraxnet-dev, FrxUSDCustodian in dev-staked-frax-erc4626		
ID	ZELLIC-T43RX6		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The targeted contracts override the `preview*` functions to skip fees when `_isFeeExempt()` returns `true`. That helper keys on the caller, not on any account passed as an argument:

```
function _isFeeExempt() internal view returns (bool _isExempt) {
  _isExempt = feeExempt[msg.sender] || feeExempt[tx.origin];
}
```

The FrxUSDCustodianBase and FrxUSDCustodian contracts build their limit functions on these previews:

```
function maxWithdraw(address _owner)
    public view returns (uint256 _maxAssetsOut) {
  uint256 _maxAssetsUser = previewRedeem(frxUSD.balanceOf(address(_owner)));
  ...
}
```

The `maxRedeem` function uses `previewWithdraw`, and `maxDeposit` uses `previewMint`. As a result, these limits use the exemption status of the caller, not of `_owner`. A fee-exempt caller sees the full pre-fee value from `maxWithdraw(_owner)`. A non-exempt caller querying the same `_owner` sees a value reduced by `redeemFee`. The NatSpec on these functions says only "Includes fee" and does not mention that the result depends on the caller.

The on-chain flow stays consistent. The `withdraw` and `redeem` functions require `_owner == msg.sender`, so the limit they check matches the caller's actual fee treatment. The mismatch appears only when a third party reads the value, for example an integrator contract, a router, or an off-chain `eth_call` made from a different `from` address.

Impact

No funds are at risk, and there is no incorrect state transition. Integrators and front ends that call `maxWithdraw`, `maxRedeem`, `maxDeposit`, or `maxMint` on behalf of a user may get a value that differs from what the user would see when calling directly. For example, a nonexempt integrator contract will see a fee-adjusted `maxWithdraw` for an exempt user's balance.

Recommendations

Document in the NatSpec of `maxWithdraw`, `maxRedeem`, `maxDeposit`, and `maxMint` that the returned values depend on the fee-exemption status of the caller (`msg.sender` or `tx.origin`). Optionally, evaluate the exemption against `_owner` inside the `max*` functions so that their results depend only on the queried account.

Remediation

This issue has been acknowledged by Frax.

3.20. Missing recipient validation lets swap output be sent to the router or executor

Target	FraxswapRouterMultihopV2	ID	ZELLIC-XG1WNJ
Category	Integration Risks	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

Note: This finding was discovered by Frax during remediation.

Both `FraxswapRouterMultihopV2.swap()` and `swapWithPermit()` pay out to `params.recipient`, or to `msg.sender` when it is `address(0)` — `_swap()` forwards this address to `FraxswapExecutorV2.execute()` without validating it:

```
address recipient = params.recipient == address(0) ? msg.sender :
    params.recipient;
// no check that the recipient is not the router or the executor
amountOut = executor.execute{ value: value }(
    params.route,
    params.amountIn,
    params.tokenOut,
    payable(recipient),
    params.amountOutMinimum
);
```

The `execute()` function measures the executor's whole balance of `tokenOut` and transfers it to `recipient`. If the recipient is the executor, the transfer goes from the executor to itself and the output stays in the contract. If the recipient is the router, ERC-20 output is sent to a contract that has no function to forward or rescue tokens.

The executor's design makes the stranded funds unrecoverable by the user. The executor documents that anything left behind is "finders keepers". Because `execute()` measures the full `tokenOut` balance, the next swapper whose route ends in the same token receives the stranded amount as part of their own output.

Impact

A swapper who sets `recipient` to the router or executor permanently loses the full swap output. In the executor case, the funds are taken by the next caller that swaps into the same `tokenOut`.

The issue needs the swapper to supply the wrong address. It can happen through an integration bug or through a malicious or compromised front end that fills in the recipient. It does not allow an attacker to take funds from a swapper who supplied a correct recipient.

Recommendations

Reject the router and the executor as recipients before running the route:

```
address recipient = params.recipient == address(0) ? msg.sender :
    params.recipient;
require(recipient != address(this) && recipient != address(executor),
    "FSR:IR"); // Invalid recipient
```

Remediation

This issue has been acknowledged by Frax, and a fix was implemented in commit 488caac2 ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Additional review observations

The review also identified an issue in legacy contracts bundled with a scoped repository, a test-evidence limitation, and a deployment constraint for the reviewed upgrade implementations. They are recorded here for completeness and do not affect the in-scope finding count.

Legacy owner sentinels can reopen initialization after ownership renunciation

Reference ID: ZELLIC-VQC7AH

The `dev-staked-frax-erc4626` repository also bundles legacy src/contracts/frxUSD/FrxUSD.sol, FrxUSD2.sol, and FrxUSD3.sol contracts outside the declared custodian paths. Their initializer uses `owner() == address(0)` as its initialization sentinel. Calling `renounceOwnership()` restores that value, allowing a later caller to initialize the proxy, become the owner, authorize itself as a minter, and mint. The bundled FrxUSD2 contract rejects a zero new owner but retains the mutable sentinel; FrxUSD3 inherits it.

A local proxy proof reproduced the takeover for the bundled FrxUSD2 and FrxUSD3 contracts. The trigger requires legacy governance to renounce ownership, but the subsequent takeover is permissionless. These contracts are outside the declared custodian scope, and the in-scope `frax-tokens` FrxUSD upgrade implementation does not expose the legacy initializer.

Use an irreversible initialization flag, and reject a zero owner. Existing proxies need an atomic migration that sets the flag before ownership can become zero; otherwise, prohibit or override `renounceOwnership`.

Reviewed implementations are upgrade-only targets

Reference ID: ZELLIC-D53ZSV

The reviewed `frax-tokens` FrxUSD and SfrxUSD implementations were evaluated for upgrade safety. Storage-layout comparisons and forked upgrade tests confirmed that they are valid upgrade targets for initialized legacy proxies.

The narrower constraint is that they are not fresh-proxy templates. The implementation instances are locked, while new empty proxies cannot reconstruct the legacy owner, timelock, token, and accounting state through a current initializer.

Document this upgrade-only assumption and require storage-layout and forked state-preservation tests for each upgrade. Fresh deployments need a separately reviewed

implementation with an atomic initializer and a constructor that locks the implementation instance.

4.2. Limited test coverage

The threat model in this report carries 2,503 coverage and negative-test markers, of which 582 are checked. Every checked marker names the supporting test function, and all 250 distinct cited test functions exist in the corresponding scoped component repository at the audited commit.

An earlier pass over the 146 markers that were checked at that time cleared 15 unsupported ones: seven relied only on audit probes or reproductions, three described RemoteMintRedeemHop effects that its scoped tests invoke but do not assert, two described oracle-cache branches that the scoped tests do not independently assert, two relied on oracle conversion tests that neither assert a non-unit price nor compare against a 1:1 baseline, and one mapped a three-argument fee-collector test to an uncalled one-argument overload. Composite state-and-event descriptions were split where tests assert the state change but not the event. One previously unchecked oracle-initialization marker was checked because `FrxUSDCustodianWithOracleTest.test_deployment` asserts its fee and mint-cap values.

The remaining 1,921 unchecked markers identify behaviors for which this review did not map a qualifying scoped-repository assertion. They do not prove that no test exists or that the behavior is defective.

Runtime execution was checked separately for the 54 test functions cited by that pass. Forty-three were observed passing: all 29 cited policy tests, all 12 cited custodian tests, the router permit-fallback test, and the Ethereum refund test. The 10 cited Tempo end-to-end tests did not reach their bodies because the suite's live-fork `setUp()` mint reverted with `PolicyForbids()`. The Arbitrum refund test could not instantiate its historical fork through the available public endpoints because archive state was unavailable. The checked markers therefore remain source-level mappings; they do not claim passes for those 11 cases.

When building a complex contract ecosystem with multiple moving parts and dependencies, comprehensive testing is essential. This includes testing for both positive and negative scenarios. Positive tests should verify that each function's side effect is as expected, while negative tests should cover every revert, preferably in every logical branch.

The test coverage for this project should be expanded to include all contracts, not just surface-level functions. It is important to test the invariants required for ensuring security and also verify mathematical properties as specified in the white paper. Additionally, testing cross-chain function calls and transfers is recommended to ensure the desired functionality.

Therefore, we recommend building a rigorous test suite that includes all contracts to ensure that the system operates securely and as intended.

Good test coverage has multiple effects.

- It finds bugs and design flaws early (preaudit or prerelease).
- It gives insight into areas for optimization (e.g., gas cost).

- It displays code maturity.
- It bolsters customer trust in your product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

The last point seems contradictory, given the time investment to create and maintain tests. To expand upon that, tests help developers trust their own changes. It is difficult to know if a code refactor — or even just a small one-line fix — breaks something if there are no tests. This is especially true for new developers or those returning to the code after a prolonged absence. Tests have your back here. They are an indicator that the existing functionality *most likely* was not broken by your change to the code.

Maintain a branch-to-test matrix for security-critical flows, prioritize the unchecked positive and negative behaviors, and keep named tests and assertions with future remediation or reassessment materials.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: CustodianWithReceiver.sol

Function: `redeemForUsdcAsync(uint256 wtgxxToRedeem)`

This function sends WTGXX from this contract to `USDC_CUSTODIAN_MAINNET` and then calls `redeemWtgxx` on it, initiating an async redemption flow. It is only callable by approved operators.

Inputs

- `wtgxxToRedeem`
  - **Control**: Fully controlled by the caller (approved operator).
  - **Constraints**: No explicit lower bound. The transfer reverts at the EVM level if it exceeds the custodian's WTGXX balance.
  - **Impact**: The amount of WTGXX transferred to `USDC_CUSTODIAN_MAINNET` and then forwarded to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS` for async redemption.

Branches and code coverage

Intended branches

- WTGXX is sent to `USDC_CUSTODIAN_MAINNET` then forwarded to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`, and `WTGXXRedeemedForUsdc` is emitted.
  - ☐ Test coverage
- The caller is a verified approved operator.
  - ☐ Test coverage

Negative behavior

- The function reverts if `msg.sender` is not an approved operator.
  - ☐ Negative test
- The function reverts if `wtgxxToRedeem` exceeds the contract's WTGXX balance.
  - ☐ Negative test

Function: `shuffleToWtgxx(uint256 amountUsdc)`

This function sends USDC held by this contract to the hardcoded `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`, intended to subscribe to WTGXX. It is only callable by `USDC_CUSTODIAN_MAINNET`.

Inputs

- `amountUsdc`
  - **Control**: Fully controlled by the caller (`USDC_CUSTODIAN_MAINNET`).
  - **Constraints**: None.
  - **Impact**: The amount of USDC transferred out of this contract to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`.

Branches and code coverage

Intended branches

- USDC is successfully transferred to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`, and `UsdcShuffledToWtgxxAsync` is emitted.
  - ☐ Test coverage
- The caller spent the specified amount of USDC from the contract's balance.
  - ☐ Test coverage

Negative behavior

- The function reverts if `msg.sender` is not `USDC_CUSTODIAN_MAINNET`.
  - ☐ Negative test
- The function reverts or behaves unexpectedly when `amountUsdc` is zero.
  - ☐ Negative test
- The function reverts when the contract has insufficient USDC balance.
  - ☐ Negative test
- The function reverts if `ON_RECEIPT_PRIMARY_ORDER_ADDRESS` is blacklisted by USDC's compliance layer.
  - ☐ Negative test

5.2. Module: CustodianWithReceiverV2.sol

Function: `_isFeeExempt()`

This function returns `true` if either `msg.sender` or `tx.origin` appears in the `feeExempt` mapping. All four `preview*` overrides gate on this.

Branches and code coverage

Intended branches

- Returns `true` when `msg.sender` is in `feeExempt`.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_previews_are_fee_free_wh`
- Returns `true` when `tx.origin` is in `feeExempt` and `msg.sender` is not.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_exempt_via_origin_throug`
- Returns `false` when neither address is exempt.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_previews_charge_fees_whe`

Function: `previewDeposit(uint256 _assetsIn)`

This function overrides the base `previewDeposit`. It applies `mintFee` only when `_isFeeExempt()` returns `false`.

Inputs

- `_assetsIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No explicit zero check.
  - **Impact**: Determines how many shares are minted.

Branches and code coverage

Intended branches

- Fee-exempt caller: the fee is zero, and the full `_assetsIn` converts to shares.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_previews_are_fee_free_wh` `FrxUSDCustodianWithReceiverV2UpgradeTest.test_deposit_exempt`)
- Nonexempt caller with nonzero `mintFee`: `_assetsIn` is reduced by the fee fraction

before conversion.

- ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_previews_charge_fees_whe`
  `FrxUSDCustodianWithRe-`
  `ceiverV2UpgradeTest.test_deposit_not_exempt`)

- Nonexempt caller with `mintFee == 0`: the result is the same as on the fee-exempt path.
  - ☐ Test coverage

Function: `previewMint(uint256 _sharesOut)`

This function overrides the base `previewMint`. It applies `mintFee` only when `_isFeeExempt()` returns `false`.

Inputs

- `_sharesOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Target number of shares — determines the assets that must be supplied.

Branches and code coverage

Intended branches

- Fee-exempt caller: no fee markup is applied to assets.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_mint_exempt`)
- Nonexempt caller: `_assetsIn` is scaled up by `1e18 / (1e18 - fee)` (ceil).
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_previews_charge_fees_whe`

Function: `previewRedeem(uint256 _sharesIn)`

This function overrides the base `previewRedeem`. It applies `redeemFee` to reduce `_assetsOut` after converting shares to assets.

Inputs

- `_sharesIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Number of shares to redeem.

Branches and code coverage

Intended branches

- Fee-exempt caller: the full asset value is returned.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_redeem_exempt`)
- Nonexempt caller: `_assetsOut = (1e18 - redeemFee) * _assetsOut / 1e18` (floor).
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_redeem_not_exempt`)

Function: `previewWithdraw(uint256 _assetsOut)`

This function overrides the base `previewWithdraw`. It applies `redeemFee` to gross up `_assetsOut`.

Inputs

- `_assetsOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Desired net asset withdrawal — determines shares required.

Branches and code coverage

Intended branches

- Fee-exempt caller: no gross-up is applied, and shares are calculated directly from `_assetsOut`.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_withdraw_exempt`)
- Nonexempt caller: `_assetsOut` is grossed up by `1e18 / (1e18 - redeemFee)` (ceil) before share conversion.
  - ☑ Test coverage

(FrxUSDCustodianWithReceiverV2UpgradeTest.test_previews_charge_fees_whe

Function: `setFeeExempt(address _user, bool _isExemptFromFee)`

This is an owner-only function that adds or removes an address from the `feeExempt` mapping.

Inputs

- `_user`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No zero-address check.
  - **Impact**: The address granted or revoked fee exemption.
- `_isExemptFromFee`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Enables or disables the fee for that address across all deposit, mint, withdraw, and redeem flows.

Branches and code coverage

Intended branches

- Grants fee exemption and emits the event.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_setFeeExempt`)
- Revokes fee exemption and emits the event.
  - ☑ Test coverage (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_setFeeExempt`, `FrxUSDCustodianWithReceiverV2UpgradeTest.test_revoked_exemption_charges_fees_again`)

Negative behavior

- Reverts when called by a non-owner.
  - ☑ Negative test (`FrxUSDCustodianWithReceiverV2UpgradeTest.test_setFeeExempt_onlyOwner`)

5.3. Module: EIP3009Lib.sol

Function: `cancelAuthorization(address authorizer, byte[32] nonce, bytes signature)`

The ASI header misrenders source `bytes32 nonce`; the declaration is external and nonpayable. Any relayer may cancel with a valid signature, while linked `DELEGATECALL` marks host storage and emits from the host.

Inputs

- `authorizer`
  - **Control**: The caller selects any address, but validation must accept the digest for that exact nonzero authorizer.
  - **Constraints**: The authorizer is signed, checked, used as the nonce-mapping key, and emitted.
  - **Impact**: It selects the EOA or ERC-1271 signer and authorization namespace.
- `nonce`
  - **Control**: The caller selects any `bytes32` nonce.
  - **Constraints**: The exact nonce must be unused and is bound into the signed cancellation digest.
  - **Impact**: It selects the sole mapping entry changed from `false` to `true`.
- `signature`
  - **Control**: The caller supplies arbitrary bytes subject to ECDSA or ERC-1271 validation.
  - **Constraints**: Malformed, rejected, or incorrect-context signatures cause `InvalidSignature()`.
  - **Impact**: It determines whether the nonce write and cancellation event occur.

Branches and code coverage

Intended branches

- Unused authorization: an unused `(authorizer, nonce)` proceeds to digest construction and signature validation.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- An accepted EOA signature marks the nonce used.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- Success emits `AuthorizationCanceled`.
  - ☐ Test coverage

- EOA validation: ECDSA recovery must resolve to the exact nonzero authorizer.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- ERC-1271 validation: the signer's `STATICCALL` must return the required ERC-1271 magic value.
  - ☐ Test coverage
- EIP-712 name from `_name`: a nonempty stored `_name` is hashed directly.
  - ☐ Test coverage
- EIP-712 legacy or empty name: an empty `_name` selects `_hashedName` or the empty-string hash.
  - ☐ Test coverage
- EIP-712 version from `_version`: a nonempty stored `_version` is hashed directly.
  - ☐ Test coverage
- EIP-712 legacy or empty version: an empty `_version` selects `_hashedVersion` or the empty-string hash.
  - ☐ Test coverage

Negative behavior

- A used authorization reverts with `UsedOrCanceledAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_CancelAuthorization_UsedOrCanceledAuthori`
- Reversion before validation is not instrumented.
  - ☐ Negative test
- Failed ECDSA validation reverts with `InvalidSignature()` and rolls back all effects.
  - ☐ Negative test
- Failed or nonconforming ERC-1271 validation reverts with `InvalidSignature()`.
  - ☐ Negative test
- A zero authorizer always reverts with `InvalidSignature()`.
  - ☐ Negative test
- Intended nonpayable wrappers reject value, while a payable host could preserve ignored outer value through `DELEGATECALL`.
  - ☐ Negative test

Function call analysis

- `this._requireUnusedAuthorization(authorizer: authorizer, nonce: nonce) -> this._getEIP3009ModuleStorage()`
    - The generated `this.*` trace misresolves private helpers; the fixed-slot Yul assignment contains no external call.
- `this._requireIsValidSignatureNow(signer: authorizer, structHash: keccak256(abi.encode(EIP3009Lib.CANCEL_AUTHORIZATION_TYPEHASH, authorizer, nonce)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: ECDSAUpgradeable.toTypedDataHash(this._domainSeparatorV4(), structHash), signature: signature)`
    - ERC-1271 validation may `STATICCALL` the authorizer before effects, with the host as caller and static context preventing mutating reentry.
- `this._requireIsValidSignatureNow(signer: authorizer, structHash: keccak256(abi.encode(EIP3009Lib.CANCEL_AUTHORIZATION_TYPEHASH, authorizer, nonce)), signature: signature) -> ECDSAUpgradeable.toTypedDataHash(this._domainSeparatorV4(), structHash)`
    - This internal hashing path binds the authorizer, nonce, name, version, chain ID, and host address without external interaction.
- `this._requireIsValidSignatureNow(signer: authorizer, structHash: keccak256(abi.encode(EIP3009Lib.CANCEL_AUTHORIZATION_TYPEHASH, authorizer, nonce)), signature: signature) -> this._domainSeparatorV4() -> this._EIP712NameHash() -> this._getEIP712Storage()`
    - These private helpers read the fixed EIP-712 slot; manual Yul inspection confirms no calls.
- `this._requireIsValidSignatureNow(signer: authorizer, structHash: keccak256(abi.encode(EIP3009Lib.CANCEL_AUTHORIZATION_TYPEHASH, authorizer, nonce)), signature: signature) -> this._domainSeparatorV4() -> this._EIP712VersionHash() -> this._getEIP712Storage()`
    - These private helpers select current, legacy, or empty version hashes without external interaction.
- `this._getEIP3009ModuleStorage()`
    - The fixed-slot helper returns host storage for the sole write, followed only by event emission.
- **Manual call-path verification**: Linked `DELEGATECALL` preserves caller and value, direct state-changing library `CALL` is guarded, storage layouts match the supplied sources, and no token/native movement or posteffect external call occurs.

Function: `validateReceiveAuthorization(address from, address to, uint256 value, uint256 validAfter, uint256 validBefore, byte[32] nonce, bytes signature)`

This ungated validator authenticates EIP-3009 authorizations, consumes the nonce, emits `AuthorizationUsed`, and returns nothing. The enclosing flow then transfers tokens, with later failures rolling back all validator effects.

Inputs

- `from`
  - **Control**: The caller supplies the candidate EOA or ERC-1271 signer.
  - **Constraints**: The `from` address must be nonzero, the `(from, nonce)` pair must be unused, and the signer must accept the signed digest.
  - **Impact**: It keys replay protection, receives signature validation, and becomes the transfer source.
- `to`
  - **Control**: The caller supplies the signed payee.
  - **Constraints**: It must equal `msg.sender`, or `InvalidPayee` is raised.
  - **Impact**: It is included in the digest and becomes the transfer recipient.
- `value`
  - **Control**: The caller supplies any signed `uint256`.
  - **Constraints**: Validation imposes no bounds, but the subsequent transfer enforces token rules.
  - **Impact**: It is signed and passed unchanged to `_transfer`.
- `validAfter`
  - **Control**: The caller supplies the signed lower time bound.
  - **Constraints**: The current `block.timestamp` must be strictly greater, or `InvalidAuthorization` is raised.
  - **Impact**: It defines the authorization's exclusive start.
- `validBefore`
  - **Control**: The caller supplies the signed upper time bound.
  - **Constraints**: The current `block.timestamp` must be strictly lower, or `ExpiredAuthorization` is raised.
  - **Impact**: It defines the authorization's exclusive expiry.
- `nonce`
  - **Control**: The caller supplies the `bytes32` value declared in the source; the generated `byte[32]` type is inaccurate.
  - **Constraints**: The `isAuthorizationUsed[from][nonce]` entry must be false.

    - **Impact**: It selects the replay-protection entry and is emitted when consumed.
- `signature`
    - **Control**: The caller supplies arbitrary signature bytes.
    - **Constraints**: EOA recovery or the ERC-1271 staticcall must accept the domain-bound digest.
    - **Impact**: It controls authentication but is not stored, emitted, or forwarded to `_transfer`.

Branches and code coverage

Intended branches

- Matching `to` and `msg.sender` binds execution to the signed payee.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- A timestamp strictly after `validAfter` passes the lower boundary.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- A timestamp strictly before `validBefore` passes the upper boundary.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- An unused `(from, nonce)` pair permits first use.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- A nonzero EOA signer accepts the tested digest.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- ERC-1271 acceptance remains untested.
    - ☐ Test coverage
- The stored, legacy, or empty EIP-712 name determines the name hash.
    - ☐ Test coverage
- The stored, legacy, or empty EIP-712 version determines the version hash.
    - ☐ Test coverage
- Success marks the nonce and dispatches `_transfer`.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- Success emits `AuthorizationUsed` before dispatching `_transfer`.

- ☐ Test coverage
- A successful enclosing call completes without return data.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)

Negative behavior

- Reachable nonpayable module entry points reject nonzero `msg.value` before delegation.
  - ☐ Negative test
- A call where `to != msg.sender` reverts first with `InvalidPayee`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidPayee_reve`
- If `block.timestamp <= validAfter`, the call reverts with `InvalidAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidAuthorizat`
- If `block.timestamp >= validBefore`, the call reverts with `ExpiredAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_ExpiredAuthorizat`
- A used or canceled nonce reverts with `UsedOrCanceledAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_UsedOrCanceledAut`
- A mismatched EOA signature reverts with `InvalidSignature`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidSignature_`
- Zero-`from` and ERC-1271 failure modes remain untested.
  - ☐ Negative test
- Reverting or malformed ERC-1271 responses reject validation before nonce consumption.
  - ☐ Negative test
- EIP-712, nonce-write, or event failures revert without persistent effects.
  - ☐ Negative test
- Transfer, balance, pause, or freeze failure rolls back the nonce and event.
  - ☐ Negative test

Function call analysis

- `this._requireUnusedAuthorization(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()`
    - Fixed-slot storage access checks replay state without an external call.
- `this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Lib.RECEIVE_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: ECDSAUpgradeable.toTypedDataHash(this._domainSeparatorV4(), structHash), signature: signature)`
    - Contract signers receive a read-only ERC-1271 `STATICCALL`; rejection prevents nonce consumption.
- `this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Lib.RECEIVE_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> ECDSAUpgradeable.toTypedDataHash(this._domainSeparatorV4(), structHash)`
    - Internal hashing binds every explicit field to the EIP-712 domain.
- `this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Lib.RECEIVE_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> this._domainSeparatorV4() -> this._EIP712NameHash() -> this._getEIP712Storage()`
    - This is fixed-slot internal access, not a literal `this.*` call or virtual hook.
- `this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Lib.RECEIVE_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> this._domainSeparatorV4() -> this._EIP712VersionHash() -> this._getEIP712Storage()`
    - The library directly reads the fixed EIP-712 version slot without invoking EIP712Lib.
- `this._markAuthorizationAsUsed(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()`
    - The fixed-slot write and event have no callback and revert atomically with downstream failure.
- **Manual call-path verification**: Linked `DELEGATECALL` preserves the caller, value, proxy, and storage. Exact Solc 0.8.22 evidence confirms that value can be preserved through a payable wrapper, although reachable entry points are nonpayable. Concrete `_transfer` resolutions have no callback. ASI failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` leave gaps only in unrelated Tempo `_quote`

paths.

Function: `validateTransferAuthorization(address from, address to, uint256 value, uint256 validAfter, uint256 validBefore, byte[32] nonce, bytes signature)`

Linked-library `DELEGATECALL` validates EIP-712 authorization, consumes the nonce, and emits `AuthorizationUsed`; wrappers then transfer tokens. The generated signature misrenders source `bytes32 nonce` as `byte[32]`.

Inputs

- `from`
  - **Control**: The relayer supplies the signer and possible ERC-1271 target.
  - **Constraints**: It must be nonzero, authorize the digest, and pass transfer checks.
  - **Impact**: It keys replay storage, appears in events, and supplies transferred tokens.
- `to`
  - **Control**: The relayer supplies the signature-bound recipient.
  - **Constraints**: Validation permits any address; transfer and FrxUSD checks may reject it.
  - **Impact**: It changes the digest and receives the subsequent token transfer.
- `value`
  - **Control**: The relayer supplies the signature-bound `uint256` value.
  - **Constraints**: Validation imposes no range check; transfer requires sufficient balance.
  - **Impact**: It changes the digest and passes unchanged into `_transfer`.
- `validAfter`
  - **Control**: The relayer supplies the signature-bound opening time.
  - **Constraints**: Success strictly requires `block.timestamp > validAfter`.
  - **Impact**: It controls authorization activation and changes the signed digest.
- `validBefore`
  - **Control**: The relayer supplies the signature-bound expiration time.
  - **Constraints**: Success strictly requires `block.timestamp < validBefore`.
  - **Impact**: It controls authorization expiration and changes the signed digest.
- `nonce`
  - **Control**: The relayer supplies the source-level `bytes32` nonce.

- **Constraints**: The `isAuthorizationUsed[from][nonce]` entry must be false.
- **Impact**: Success marks and emits the pair, subject to transaction rollback.

- `signature`
  - **Control**: The relayer supplies arbitrary bytes or packed `r || s || v`.
  - **Constraints**: ECDSA or ERC-1271 must accept the exact typed-data digest.
  - **Impact**: Acceptance permits nonce consumption and the subsequent transfer.

Branches and code coverage

Intended branches

- Both strict time-window checks pass.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- The `(from, nonce)` authorization remains unused.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- ECDSA recovery validates nonzero `from`.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- ERC-1271 fallback returns the required magic value.
  - ☐ Test coverage
- The domain selects stored `_name`, legacy `_hashedName`, or empty fallback.
  - ☐ Test coverage
- The domain selects stored `_version`, legacy `_hashedVersion`, or empty fallback.
  - ☐ Test coverage
- Success marks the nonce used.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Success emits `AuthorizationUsed`.
  - ☐ Test coverage
- The module resolves `_transfer` to ERC-20 accounting and moves the tested balance.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- FrxUSD owner relayers bypass hooks; others pass pause and freeze checks.
  - ☐ Test coverage

Negative behavior

- If `block.timestamp <= validAfter`, the call reverts with `InvalidAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidAuthoriza`
- If `block.timestamp >= validBefore`, the call reverts with `ExpiredAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_ExpiredAuthoriza`
- Used or canceled nonces revert with `UsedOrCanceledAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_UsedOrCanceledAu`
- A valid EOA signature over different transfer fields reverts with `InvalidSignature`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidSignature`
- Reverting, malformed, or wrong-magic ERC-1271 validation remains untested.
  - ☐ Negative test
- If `from == address(0)`, the call reverts with `InvalidSignature`.
  - ☐ Negative test
- Exceptional hashing or signature-checking failure precedes nonce consumption.
  - ☐ Negative test
- Nonpayable wrappers reject value, though `DELEGATECALL` preserves it.
  - ☐ Negative test
- Later transfer or hook failure rolls back nonce and event changes.
  - ☐ Negative test
- FrxUSD blocks non-owner relayers when paused or relevant addresses are frozen.
  - ☐ Negative test

Function call analysis

- `this._requireUnusedAuthorization(...) -> this._getEIP3009ModuleStorage()`
  - The generated `this.*` trace misresolves private helpers; the fixed-slot Yul assignment contains no external call.
- `this._requireIsValidSignatureNow(...) -> SignatureChecker.isValidSignatureNow(...)`
  - ECDSA may short-circuit. Otherwise, ERC-1271 `STATICCALL` accepts only valid magic data and maps ordinary failures to `InvalidSignature`. It can still exhaust gas.

- `this._requireIsValidSignatureNow(...) -> this._domainSeparatorV4() -> this._EIP712NameHash() -> this._getEIP712Storage()`
    - These private helpers read the fixed EIP-712 slot; manual Yul inspection confirms no calls.
- `this._requireIsValidSignatureNow(...) -> this._domainSeparatorV4() -> this._EIP712VersionHash() -> this._getEIP712Storage()`
    - These private helpers select current, legacy, or empty version hashes without external interaction.
- `this._markAuthorizationAsUsed(...) -> this._getEIP3009ModuleStorage()`
    - The fixed-slot helper returns host storage for the sole write, followed only by event emission.
- **Manual call-path verification**: The flow is nonpayable wrapper → linked-library `DELEGATECALL` → private helpers → optional ERC-1271 `STATICCALL` → nonce/event → ERC-20 transfer and FrxUSD hook. Constant-slot Yul helpers make no external calls; `this.*` labels are inaccurate. Solc 0.8.22 confirmed value preservation 2/2, direct library calls hit the runtime guard, and unrelated Tempo `_payNative` / fee-routing builder errors do not affect this path.

5.4. Module: EIP3009Module.sol

Function: `cancelAuthorization(address authorizer, byte[32] nonce, uint8 v, byte[32] r, byte[32] s)`

The heading uses `byte[32]`, while the source uses `bytes32`. Any account may relay this nonpayable wrapper, which packs `r || s || v`, authenticates the domain-bound cancellation, marks the shared nonce used, and emits `AuthorizationCanceled`.

Inputs

- `authorizer`
    - **Control**: Any relayer supplies the claimed signer and ERC-1271 target.
    - **Constraints**: Success requires this address to authenticate the domain-separated cancellation digest.
    - **Impact**: Selects the shared nonce namespace, mapping key, and indexed event value.
- `nonce`
    - **Control**: The relayer supplies this signature-bound mapping key.
    - **Constraints**: The selected entry must be unused, with no format, ordering, or deadline constraint.
    - **Impact**: Success prevents later transfer, receive, or cancellation with the

same authorizer and nonce.

- `v`
  - **Control**: The relayer supplies the final byte of the packed signature.
  - **Constraints**: ABI decoding enforces `uint8`, while signature validation determines acceptance.
  - **Impact**: A rejected value causes `InvalidSignature()` without persistent effects.
- `r`
  - **Control**: The relayer supplies bytes 1–32 of the packed signature.
  - **Constraints**: ECDSA recovery or the authorizer's ERC-1271 policy determines acceptance.
  - **Impact**: A rejected value prevents the nonce write and event.
- `s`
  - **Control**: The relayer supplies bytes 33–64 of the packed signature.
  - **Constraints**: OpenZeppelin ECDSA rules or the authorizer's ERC-1271 policy determine acceptance.
  - **Impact**: A rejected value prevents the nonce write and event.

Branches and code coverage

Intended branches

- The nonpayable entry accepts zero value from any relayer.
  - ☐ Test coverage
- The wrapper packs exactly `r || s || v` and internally dispatches with unchanged arguments.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- A used nonce reverts with `UsedOrCanceledAuthorization`.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_UsedOrCanceledAuthori`
- The test does not instrument whether hashing or signature validation ran first.
  - ☐ Test coverage
- FrxUSD uses OpenZeppelin EIP-712 hashing, while SfrxUSD uses its dynamic domain separator.
  - ☐ Test coverage
- Successful ECDSA recovery short-circuits ERC-1271 validation.
  - ☑ Test coverage

(FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds)

- Failed ECDSA authentication conditionally triggers a valueless ERC-1271 staticcall.
  - ☐ Test coverage
- Successful ECDSA authentication writes the authorization state to true.
  - ☑ Test coverage (FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds)
- Success emits AuthorizationCanceled and returns no data.
  - ☐ Test coverage
- No expiry, caller-role, token-transfer, accounting, or callback branch applies.
  - ☐ Test coverage

Negative behavior

- Nonzero msg.value is rejected before body execution.
  - ☐ Negative test
- A previously used nonce raises UsedOrCanceledAuthorization().
  - ☑ Negative test (FraxOFTUpgradeableTest.test_CancelAuthorization_UsedOrCanceledAuthori
- Post-revert state is not independently asserted.
  - ☐ Negative test
- A signature invalid for the exact action and domain raises InvalidSignature().
  - ☐ Negative test
- ERC-1271 failure, short data, or incorrect magic value becomes InvalidSignature().
  - ☐ Negative test
- Static context prevents the signer from completing state-changing reentry.
  - ☐ Negative test
- Replaying or later using the canceled nonce raises UsedOrCanceledAuthorization().
  - ☑ Negative test (FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds)
- Signatures bound to another signer, nonce, action, token, chain, or domain are rejected.
  - ☐ Negative test

Function call analysis

- this.cancelAuthorization(authorizer: authorizer, nonce: nonce, signature: abi.encodePacked(r, s, v)) -> this._requireUnusedAuthorization(authorizer: authorizer, nonce: nonce)

`-> this._getEIP3009ModuleStorage()`

    - The `this.*` rendering mislabels internal calls, and the Yul helper only assigns the fixed storage slot.
- `this.cancelAuthorization(authorizer: authorizer, nonce: nonce, signature: abi.encodePacked(r, s, v)) -> this._requireIsValidSignatureNow(signer: authorizer, structHash: keccak256(abi.encode(EIP3009Module.CANCEL_AUTHORIZATION_TYPEHASH, authorizer, nonce)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: this.__hashTypedDataV4(structHash: structHash), signature: signature)`
    - These calls are internal, but failed ECDSA authentication conditionally causes the sole external edge: a read-only ERC-1271 `staticcall` to `authorizer` before effects.
- `this.cancelAuthorization(authorizer: authorizer, nonce: nonce, signature: abi.encodePacked(r, s, v)) -> this._getEIP3009ModuleStorage()`
    - This internal fixed-slot access performs the authenticated write and is followed only by event emission.
- **Manual call-path verification**: The path is wrapper → unused check → concrete domain hashing → ECDSA or conditional ERC-1271 validation → nonce write → event, with no external self-call or post-write interaction.

Function: `cancelAuthorization(address authorizer, byte[32] nonce, bytes signature)`

Any relayer may cancel an unused `(authorizer, nonce)` with a valid domain-separated signature; success sets the used bit and emits `AuthorizationCanceled` without moving assets. The generated `byte[32]` differs from Solidity's `bytes32`; minter-removal gas, zero-address freezing, and timelock-delay assumptions are outside this path.

Inputs

- `authorizer`
    - **Control**: The caller selects the authorizer, which may be an EOA or ERC-1271 contract.
    - **Constraints**: Its nonce must be unused, and it must validate the exact domain-separated digest.
    - **Impact**: It selects the mapping namespace, signer, callback target, and indexed event field.
- `nonce`
    - **Control**: The caller supplies all 32 bytes without transformation.
    - **Constraints**: It must be unused for the selected authorizer and is

cryptographically bound into the signed struct.
    - **Impact**: Success permanently marks the selected pair as used unless future upgrade code changes storage.
- `signature`
    - **Control**: The caller supplies arbitrary bytes, while the alternate overload packs caller-controlled `r`, `s`, and `v`.
    - **Constraints**: It must recover the authorizer or produce valid ERC-1271 magic-value return data.
    - **Impact**: It gates the write and event but is neither stored nor emitted.
- The caller (`msg.sender`) only relays the request, and `tx.origin` is unused. The call value (`msg.value`) must be zero, and the proxy context determines storage and the verifying-contract domain.
- FrxUSD and sfrxUSD use distinct chain- and address-bound domain-hash implementations.

Branches and code coverage

Intended branches

- The internally reached bytes-signature path sets an unused authorization bit.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- The successful path emits `AuthorizationCanceled`.
    - ☐ Test coverage
- The `(v, r, s)` overload packs `r || s || v` and dispatches internally with unchanged authorizer and nonce.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- A matching EOA signature succeeds through ECDSA recovery without an external call.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_CancelAuthorization_succeeds`)
- A contract authorizer succeeds when its ERC-1271 staticcall returns valid magic-value data.
    - ☐ Test coverage
- FrxUSD resolves domain hashing through `EIP712._hashTypedDataV4`.
    - ☐ Test coverage
- sfrxUSD resolves domain hashing using its prefix, `DOMAIN_SEPARATOR()`, and `computeDomainSeparator()`.
    - ☐ Test coverage

Negative behavior

- An already used authorization reverts with `UsedOrCanceledAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_CancelAuthorization_UsedOrCanceledAuthori`
- Reversion before signature checking is not instrumented.
  - ☐ Negative test
- Failed EOA recovery followed by failed ERC-1271 validation reverts with `InvalidSignature()`.
  - ☐ Negative test
- ERC-1271 revert, failed call, short data, or incorrect magic value causes `InvalidSignature()` with no retained effects.
  - ☐ Negative test
- A signature for another chain, contract, token domain, authorizer, or nonce fails validation.
  - ☐ Negative test
- A later transaction loses any consume-or-cancel race and reverts after observing the used bit.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by nonpayable dispatch before execution.
  - ☐ Negative test
- Any relayer may submit a valid cancellation because neither `msg.sender` nor `tx.origin` authorizes it.
  - ☐ Negative test
- A malicious ERC-1271 signer may revert, consume gas, or make read-only nested calls, but static context prevents persistent reentrancy.
  - ☐ Negative test

Function call analysis

- `this._requireUnusedAuthorization(authorizer: authorizer, nonce: nonce) -> this._getEIP3009ModuleStorage()`
  - These are internal fixed-slot operations despite the misleading `this.*` rendering; if the bit is true, the function reverts before any external interaction or effects.
- `this._requireIsValidSignatureNow(signer: authorizer, structHash: keccak256(abi.encode(EIP3009Module.CANCEL_AUTHORIZATION_TYPEHASH, authorizer, nonce)), signature: signature) ->`

```
SignatureChecker.isValidSignatureNow(signer: signer, hash:
this.__hashTypedDataV4(structHash: structHash), signature: signature)
```

    - EOA recovery occurs internally. If it fails, the function makes the only external call: a valueless ERC-1271 `STATICCALL` to `authorizer` before any write.
- `this._getEIP3009ModuleStorage()`
    - This internal assembly helper returns the constant storage slot used to set only `isAuthorizationUsed[authorizer][nonce]`.
- **Manual call-path verification**: Runtime order is nonpayable checking, unused-bit read, concrete domain hashing, EOA or ERC-1271 validation, bit write, and event emission. FCA renders internal helpers as external calls, conflates domain overrides, and omits the conditional ERC-1271 edge. No transfer, approval, delegatecall, creation, or postwrite call occurs.

Function: `receiveWithAuthorization(address from, address to, uint256 value, uint256 validAfter, uint256 validBefore, byte[32] nonce, bytes signature)`

Signed payees submit timely, unused EIP-3009 authorizations for exact transfers. The source uses `bytes32 nonce`. A revert undoes nonce consumption, events, and transfer effects.

Inputs

- `from`
    - **Control**: Caller-supplied payer authenticated by ECDSA or ERC-1271.
    - **Constraints**: Its nonce must be unused, its signature valid, and its balance and policy checks satisfied.
    - **Impact**: Selects the signer, nonce namespace, debited account, and `AuthorizationUsed` authorizer.
- `to`
    - **Control**: Caller-supplied payee bound to the signature.
    - **Constraints**: It must equal `msg.sender` and satisfy the concrete transfer policy.
    - **Impact**: Selects the credited account and prevents third-party submission.
- `value`
    - **Control**: Caller-supplied amount bound to the signature.
    - **Constraints**: Zero is allowed, while insufficient balance reverts.
    - **Impact**: Determines the token or share amount transferred.
- `validAfter`

    - **Control**: Caller-supplied opening time bound to the signature.
    - **Constraints**: The current `block.timestamp` must be strictly greater than it.
    - **Impact**: Defines the authorization's exclusive opening boundary.
- `validBefore`
    - **Control**: Caller-supplied closing time bound to the signature.
    - **Constraints**: The current `block.timestamp` must be strictly less than it.
    - **Impact**: Defines the authorization's exclusive closing boundary.
- `nonce`
    - **Control**: Caller-supplied `bytes32` bound to the signature.
    - **Constraints**: The `isAuthorizationUsed[from][nonce]` entry must be false across the shared authorization namespace.
    - **Impact**: Selects the consumed mapping entry and emitted indexed nonce.
- `signature`
    - **Control**: Arbitrary bytes or packed `r || s || v` from the forwarding overload.
    - **Constraints**: It must validate for `from` through ECDSA or ERC-1271.
    - **Impact**: Gates nonce consumption and transfer without being stored or emitted.

Branches and code coverage

Intended branches

- A call where `to == msg.sender` permits time validation, while internal overload forwarding preserves the caller.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- A `block.timestamp` greater than `validAfter` satisfies the strict opening boundary.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- A `block.timestamp` less than `validBefore` satisfies the strict closing boundary.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- An unused `isAuthorizationUsed[from][nonce]` permits signature validation.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- Valid ECDSA recovery authenticates `from`.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)

- Failed ECDSA may fall back to an ERC-1271 `staticcall` whose magic value authenticates `from`.
  - ☐ Test coverage
- Successful validation marks the nonce used.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- Successful validation emits `AuthorizationUsed`.
  - ☐ Test coverage
- FrxUSD applies pause, freeze, balance, and ERC-20 accounting before returning `true`.
  - ☐ Test coverage
- SfrxUSD debits and credits shares, emits `Transfer`, and returns `true`.
  - ☐ Test coverage
- A zero-value authorization may succeed and consume its nonce.
  - ☐ Test coverage

Negative behavior

- A call where `to != msg.sender` reverts with `InvalidPayee`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidPayee_reve`
- If `block.timestamp <= validAfter`, the call reverts with `InvalidAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidAuthorizat`
- If `block.timestamp >= validBefore`, the call reverts with `ExpiredAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_ExpiredAuthorizat`
- A used or canceled nonce reverts with `UsedOrCanceledAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_UsedOrCanceledAut`
- A valid EOA signature over a different `to` value fails with `InvalidSignature`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidSignature_`
- ERC-1271 fallback behavior remains untested.
  - ☐ Negative test
- Reverting, malformed, short, or non-magic ERC-1271 responses produce `InvalidSignature`.
  - ☐ Negative test

- FrxUSD pause, freeze, balance, or ERC-20 policy failures revert unless an applicable owner bypass exists.
  - ☐ Negative test
- SfrxUSD insufficient balance reverts on checked subtraction.
  - ☐ Negative test
- Any transfer revert rolls back nonce state, events, and transfer effects.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable dispatcher.
  - ☐ Negative test

Function call analysis

- this._requireUnusedAuthorization(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()
  - The fixed-slot Yul helper only reads namespaced nonce storage and cannot call externally.
- this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Module.RECEIVE_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: this.__hashTypedDataV4(structHash: structHash), signature: signature)
  - Failed ECDSA may `staticcall` caller-selected `from`. Only ERC-1271 magic authenticates it. The static context prevents state-changing reentrancy.
- this._markAuthorizationAsUsed(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()
  - The fixed-slot helper writes the authenticated nonce and emits `AuthorizationUsed` without external interaction.
- `__transfer({ from: from, to: to, amount: value })` — source call omitted from the generated FCA paths.
  - FrxUSD uses internal `ERC20._transfer`. sfrxUSD updates shares directly. Neither invokes callbacks or other contracts.
- **Manual call-path verification**: Generated `this.*` edges are internal, and slot Yul makes no call. ERC-1271 validation is the only runtime external edge. If a later transfer reverts, nonce and event state revert with it.

Findings

- No additional EIP-3009 issue was found. Both concrete `__transfer` implementations return `true` or revert. Domain separation uses OpenZeppelin EIP-712 or sfrxUSD's

runtime separator.
- Zero-address freeze and mint-burn liveness, minter administration, and timelock authentication are outside this path. The generated trace also renders `bytes32` as `byte[32]`, shows internal calls as `this.*`, flags noncalling slot Yul, and omits `__transfer`.

Function: `receiveWithAuthorization(address from, address to, uint256 value, uint256 validAfter, uint256 validBefore, byte[32] nonce, uint8 v, byte[32] r, byte[32] s)`

This permissionless wrapper packs `r || s || v` and internally invokes the `bytes` overload while preserving `msg.sender`. Authorization is consumed before transfer, and failures revert atomically.

Inputs

- `from`
  - **Control**: The caller supplies the signer and debited account.
  - **Constraints**: Its nonce must be unused and its signature valid.
  - **Impact**: It selects the authorization namespace and debited balance.
- `to`
  - **Control**: The caller supplies the signed recipient.
  - **Constraints**: It must equal `msg.sender` and satisfy applicable freeze rules.
  - **Impact**: It identifies the authorized submitter and credited account.
- `value`
  - **Control**: The caller supplies the signed transfer amount.
  - **Constraints**: Sufficient balance and applicable compliance checks are required.
  - **Impact**: It determines the balance movement and `Transfer` amount.
- `validAfter`
  - **Control**: The caller supplies the signed lower time bound.
  - **Constraints**: The current `block.timestamp` must be strictly greater than this value.
  - **Impact**: It establishes the authorization's exclusive start time.
- `validBefore`
  - **Control**: The caller supplies the signed upper time bound.
  - **Constraints**: The current `block.timestamp` must be strictly less than this value.
  - **Impact**: It establishes the authorization's exclusive expiry time.
- `nonce`

- **Control**: The caller supplies the `bytes32` nonce declared in source.
- **Constraints**: The `(from, nonce)` entry must be unused and uncanceled.
- **Impact**: It selects the consumed replay-protection entry.

- `v`
  - **Control**: The caller supplies the final signature byte.
  - **Constraints**: It must produce an accepted ECDSA or ERC-1271 signature.
  - **Impact**: It affects signature acceptance only.
- `r`
  - **Control**: The caller supplies the `bytes32` signature component declared in source.
  - **Constraints**: It is packed unchanged into signature bytes 0–31.
  - **Impact**: It affects signature acceptance only.
- `s`
  - **Control**: The caller supplies the `bytes32` signature component declared in source.
  - **Constraints**: It is packed unchanged into signature bytes 32–63.
  - **Impact**: It affects signature acceptance only.

Branches and code coverage

Intended branches

- The wrapper packs `r || s || v` and internally calls the `bytes` overload.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- Valid payee, time, nonce, and signature checks lead to nonce consumption and the expected transfer.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- Exact ordering of nonce consumption before transfer accounting is not instrumented.
  - ☐ Test coverage
- Valid EOA signatures recover to `from`.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- Failed ECDSA recovery may use ERC-1271, requiring its magic value.
  - ☐ Test coverage
- FrxUSD3 non-owner transfers enforce pause and freeze checks.
  - ☐ Test coverage

- FrxUSD3 owner callers bypass compliance checks but still require sufficient balance.
  - ☐ Test coverage
- SfrxUSD3 debits, credits, emits `Transfer`, and returns `true`.
  - ☐ Test coverage
- Zero-value authorizations consume their nonce and emit both events when valid.
  - ☐ Test coverage
- The concrete transfer hook completes ERC-20 balance movement.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_succeeds`)
- The hook return value is not asserted.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` is rejected by the nonpayable dispatcher.
  - ☐ Negative test
- A `to` differing from `msg.sender` reverts with `InvalidPayee`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidPayee_reve`
- A timestamp at or before `validAfter` reverts with `InvalidAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidAuthorizat`
- A timestamp at or after `validBefore` reverts with `ExpiredAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_ExpiredAuthorizat`
- A used nonce reverts with `UsedOrCanceledAuthorization`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_UsedOrCanceledAut`
- Reversion before signature validation is not instrumented.
  - ☐ Negative test
- A mismatched EOA signature reverts with `InvalidSignature`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_InvalidSignature_`
- Reverting, malformed, or wrong-magic ERC-1271 validation remains untested.
  - ☐ Negative test
- Reverting, malformed, short, or nonmagic ERC-1271 responses fail validation.

- ☐ Negative test
- FrxUSD3 pause or freeze failures revert atomically.
  - ☐ Negative test
- Insufficient balance reverts atomically.
  - ☐ Negative test
- Reusing a consumed or canceled nonce reverts.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_ReceiveWithAuthorization_UsedOrCanceledAu`

Function call analysis

- `this.receiveWithAuthorization(...) -> this._requireUnusedAuthorization(from, nonce) -> this._getEIP3009ModuleStorage()`
- `this.receiveWithAuthorization(...) -> this._requireIsValidSignatureNow(...) -> SignatureChecker.isValidSignatureNow(...)`
  - Failed ECDSA recovery may `staticcall` ERC-1271; invalid responses fail before nonce marking.
- `this.receiveWithAuthorization(...) -> this._markAuthorizationAsUsed(from, nonce) -> this._getEIP3009ModuleStorage()`
- **Source call omitted by generated FCA — `FrxUSD3.__transfer(...) -> ERC20._transfer(...) -> FrxUSD2._update(...) -> ERC20._update(...)`**: Internal accounting enforces compliance and balance rules without recipient hooks.
- **Source call omitted by generated FCA — `SfrxUSD3.__transfer(...)`**: Internal accounting emits `Transfer` without external calls.
- **Manual call-path verification**: Generated `this.` notation denotes internal calls, and concrete transfer dispatch was manually confirmed.

Findings

This transfer path does not use freezer administration, minter removal, or timelock configuration. It remains nonce-safe, time-bounded, domain-separated, and limited to transfers returning `true` or reverting; FrxUSD3's owner compliance bypass is explicit authority.

Function: `transferWithAuthorization(address from, address to, uint256 value, uint256 validAfter, uint256 validBefore, byte[32] nonce, uint8 v, byte[32] r, byte[32] s)`

Relayers submit packed `r || s || v` authorization while `msg.sender` remains unchanged. Success consumes the nonce, emits `AuthorizationUsed` before `Transfer`, and returns nothing.

Inputs

- `from`
  - **Control**: Calldata selects the signer, requiring valid ECDSA or ERC-1271 authorization.
  - **Constraints**: It keys nonce state and must satisfy concrete transfer checks.
  - **Impact**: It selects the signer, debit account, nonce namespace, and ERC-1271 target.
- `to`
  - **Control**: The relayer supplies the recipient, which the signer authorizes.
  - **Constraints**: frxUSD rejects zero or frozen recipients; sfrxUSD permits zero.
  - **Impact**: It selects the credited account and `Transfer` recipient.
- `value`
  - **Control**: The relayer supplies an arbitrary signed `uint256` value.
  - **Constraints**: Zero is allowed, no allowance is spent, and balance must suffice.
  - **Impact**: It determines the exact token or share balance movement.
- `validAfter`
  - **Control**: The relayer supplies the signed lower timestamp bound.
  - **Constraints**: Success requires `block.timestamp > validAfter`.
  - **Impact**: Failure causes `InvalidAuthorization()` before nonce or signature processing.
- `validBefore`
  - **Control**: The relayer supplies the signed upper timestamp bound.
  - **Constraints**: Success requires `block.timestamp < validBefore`.
  - **Impact**: Failure causes `ExpiredAuthorization()` before nonce consumption.
- `nonce`
  - **Control**: The relayer supplies the signed 32-byte nonce.
  - **Constraints**: The shared `(from, nonce)` entry must be unused before success.
  - **Impact**: It prevents replay across transfer, receive, and cancellation paths.

- `v`
  - **Control**: The relayer supplies the signature's final byte.
  - **Constraints**: It is appended to `r || s` without wrapper validation.
  - **Impact**: It affects only ECDSA or ERC-1271 signature acceptance.
- `r`
  - **Control**: The relayer supplies the signature's first 32 bytes.
  - **Constraints**: Curve-range validation is delegated to the signature checker.
  - **Impact**: It affects recovery or the ERC-1271 signature payload.
- `s`
  - **Control**: The relayer supplies the signature's middle 32 bytes.
  - **Constraints**: Low-`s` and curve validation are delegated to the signature checker.
  - **Impact**: It affects signature acceptance without directly changing balances.

Branches and code coverage

Intended branches

- The nonpayable wrapper accepts zero native value and preserves external `msg.sender`.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- The wrapper packs `r || s || v` and calls the bytes-signature overload internally.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Time validation requires `validAfter < block.timestamp < validBefore`.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- An unused shared `(from, nonce)` permits signature validation.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- frxUSD uses EIP-712 hashing; sfrxUSD directly combines domain and struct hashes.
  - ☐ Test coverage
- Validation accepts the tested ECDSA signature without ERC-1271 fallback.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Validation can alternatively accept ERC-1271's returned magic value.
  - ☐ Test coverage

- Successful validation consumes the tested nonce and completes transfer accounting.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Nonce-consumption ordering relative to transfer accounting is not instrumented.
  - ☐ Test coverage
- frxUSD checks endpoints, balances, pause, and freezes, with owner-relayer bypasses.
  - ☐ Test coverage
- sfrxUSD uses checked debit, unchecked credit, and permits zero recipients.
  - ☐ Test coverage
- The concrete transfer hook completes ERC-20 balance movement.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- The hook return value is not asserted.
  - ☐ Test coverage
- Success emits `AuthorizationUsed` before `Transfer` and returns no ABI data.
  - ☐ Test coverage

Negative behavior

- A timestamp equal to `validAfter` reverts with `InvalidAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidAuthoriza`
- Ordering before authorization-state access is not instrumented.
  - ☐ Negative test
- A timestamp equal to `validBefore` reverts with `ExpiredAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_ExpiredAuthoriza`
- Ordering before nonce consumption is not instrumented.
  - ☐ Negative test
- A used nonce reverts with `UsedOrCanceledAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_UsedOrCanceledAu`
- Reversion before signature interaction is not instrumented.
  - ☐ Negative test
- A mismatched EOA signature reverts with `InvalidSignature()` without marking the nonce.

- ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidSignature`)
- Reverting, malformed, or wrong-magic ERC-1271 validation remains untested.
  - ☐ Negative test
- Reverting, short, or nonmagic ERC-1271 responses fail validation.
  - ☐ Negative test
- frxUSD rejects zero endpoints, insufficient balances, and applicable compliance violations.
  - ☐ Negative test
- sfrxUSD reverts on debit underflow but permits a zero recipient.
  - ☐ Negative test
- Later transfer failure restores the nonce and removes both logs.
  - ☐ Negative test
- Nonzero `msg.value` is rejected before the nonpayable wrapper executes.
  - ☐ Negative test

Function call analysis

- `this.transferWithAuthorization(from: from, to: to, value: value, validAfter: validAfter, validBefore: validBefore, nonce: nonce, signature: abi.encodePacked(r, s, v)) -> this._requireUnusedAuthorization(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()`
  - A fixed-slot read of the host's nonce mapping, with no external call.
- `this.transferWithAuthorization(from: from, to: to, value: value, validAfter: validAfter, validBefore: validBefore, nonce: nonce, signature: abi.encodePacked(r, s, v)) -> this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Module.TRANSFER_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: this.__hashTypedDataV4(structHash: structHash), signature: signature)`
  - The only external edge is an ERC-1271 `staticcall`; failure or non-magic output causes `InvalidSignature()`.
- **Manual call-path verification**: Yul helpers only assign a fixed slot; ASI's `this.*` notation misrenders internal calls and omits virtual transfer dispatch; `byte[32]` differs from source `bytes32`; these discrepancies do not create an authorization bypass.

Function: `transferWithAuthorization(address from, address to, uint256 value, uint256 validAfter, uint256 validBefore, byte[32] nonce, bytes signature)`

Any caller may relay a signed EIP-3009 transfer subject to strict time bounds, concrete-domain signature validity, and an unused authorizer nonce. The source ABI uses `bytes32`, not the ASI-rendered `byte[32]`, and success marks the nonce before token accounting.

Inputs

- `from`
  - **Control**: The relayer supplies `from`, but the signature must validate for that address.
  - **Constraints**: The nonce must be unused, the signature valid, and the concrete transfer able to debit `from`.
  - **Impact**: It selects the signer, ERC-1271 target, nonce namespace, event authorizer, and debited balance.
- `to`
  - **Control**: The relayer supplies `to`, and the signature binds it exactly.
  - **Constraints**: frxUSD always rejects zero recipients and rejects frozen recipients for non-owner relayers, while sfrxUSD permits zero and self recipients.
  - **Impact**: It selects the credited balance and `Transfer` recipient without receiving Ether or underlying assets.
- `value`
  - **Control**: The relayer supplies the full signed `uint256` value.
  - **Constraints**: Sufficient sender balance is required, while zero-value transfers are permitted.
  - **Impact**: It determines own-token balance movement without changing allowance, underlying assets, or total supply.
- `validAfter`
  - **Control**: The relayer supplies the signer-bound lower timestamp.
  - **Constraints**: Execution requires `block.timestamp > validAfter`.
  - **Impact**: It establishes the authorization's exclusive lower time bound.
- `validBefore`
  - **Control**: The relayer supplies the signer-bound upper timestamp.
  - **Constraints**: Execution requires `block.timestamp < validBefore`.
  - **Impact**: It establishes the authorization's exclusive upper time bound.
- `nonce`
  - **Control**: The relayer supplies the signer-bound `bytes32` nonce.

    - **Constraints**: The `isAuthorizationUsed[from][nonce]` entry must be false across transfer, receive, and cancellation paths.
    - **Impact**: It selects the replay-protection entry marked and emitted before token accounting.
- `signature`
    - **Control**: The caller supplies arbitrary bytes, or the `(v,r,s)` overload packs `r || s || v`.
    - **Constraints**: ECDSA recovery or ERC-1271 validation for `from` must succeed against the concrete EIP-712 domain.
    - **Impact**: It authorizes this transfer but is neither stored nor emitted.

Branches and code coverage

Intended branches

- Execution passes the lower bound only when `block.timestamp > validAfter`.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Execution passes the upper bound only when `block.timestamp < validBefore`.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- An unused `isAuthorizationUsed[from][nonce]` permits signature validation.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- A valid ECDSA signature recovering `from` passes without ERC-1271.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Failed ECDSA recovery may fall back to a valid ERC-1271 response from `from`.
    - ☐ Test coverage
- The signature binds the tested `to` field rather than the relayer.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidSignature`
- A caller distinct from `from` relays the `(v,r,s)` overload, which reaches the signed transfer path.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)
- Success marks the nonce and completes token accounting.
    - ☑ Test coverage (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_succeeds`)

- Success emits `AuthorizationUsed` before token accounting emits `Transfer`.
  - ☐ Test coverage
- frxUSD applies non-owner pause and freeze checks plus ERC-20 address and balance checks.
  - ☐ Test coverage
- sfrxUSD uses checked debit and unchecked credit while permitting zero and self recipients.
  - ☐ Test coverage
- Both concrete hooks return `true` after transfer or revert, while the outer function returns no data.
  - ☐ Test coverage

Negative behavior

- If `block.timestamp <= validAfter` (including equality), the call reverts with `InvalidAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidAuthoriza`
- If `block.timestamp >= validBefore` (including equality), the call reverts with `ExpiredAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_ExpiredAuthoriza`
- If `validAfter >= validBefore`, no timestamp satisfies both strict bounds.
  - ☐ Negative test
- A used or canceled nonce reverts with `UsedOrCanceledAuthorization()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_UsedOrCanceledAu`
- A valid EOA signature over different transfer fields reverts with `InvalidSignature()`.
  - ☑ Negative test (`FraxOFTUpgradeableTest.test_TransferWithAuthorization_InvalidSignature`
- ERC-1271 fallback failure modes remain untested.
  - ☐ Negative test
- ERC-1271 failure, malformed return data, or a nonmagic value results in `InvalidSignature()`.
  - ☐ Negative test
- frxUSD pause or freeze failure rolls back tentative nonce use and events.
  - ☐ Negative test
- frxUSD zero-address or insufficient-balance failure rolls back nonce consumption.

- ☐ Negative test

- sfrxUSD insufficient balance panics on checked subtraction and rolls back nonce consumption.
  - ☐ Negative test
- Any later revert atomically restores authorization and token state and discards events.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by nonpayable dispatch.
  - ☐ Negative test

Function call analysis

- `this._requireUnusedAuthorization(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()`
  - These are internal fixed-slot nonce operations despite ASI's `this.*` rendering.
- `this._requireIsValidSignatureNow(signer: from, structHash: keccak256(abi.encode(EIP3009Module.TRANSFER_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter, validBefore, nonce)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: this.__hashTypedDataV4(structHash: structHash), signature: signature)`
  - The only external edge is SignatureChecker's ERC-1271 `staticcall` to caller-selected `from` after ECDSA fails, with no stateful reentrancy.
- `this._markAuthorizationAsUsed(authorizer: from, nonce: nonce) -> this._getEIP3009ModuleStorage()`
  - These internal operations mark the fixed nonce slot and emit `AuthorizationUsed` before the concrete transfer.
- **Manual call-path verification**: ASI omits concrete transfer and domain overrides and misrenders internal calls; actual execution performs strict time checks, nonce read, concrete-domain signature validation, nonce marking, and then internal frxUSD or sfrxUSD accounting, while the `(v,r,s)` overload dispatches internally unchanged.

5.5. Module: FraxOFTAdapterUpgradeable.sol

Function: `checkpointRateLimits(uint32[] _eids)`

This owner-only wrapper forwards EIDs to the linked RateLimiterLib; its `DELEGATECALL` updates namespaced storage and emits per-entry events in the current execution context.

Inputs

- `_eids`
  - **Control**: The owner controls array length, order, duplicates, and each `uint32` EID.
  - **Constraints**: Valid ABI encoding is required, with no nonempty, allowlist, sorting, deduplication, or explicit length requirement.
  - **Impact**: Entries are processed in order using per-EID overrides or the default configuration.
- `msg.sender`
  - **Control**: The transaction or immediate calling contract determines `msg.sender`.
  - **Constraints**: The `onlyOwner` check requires it to equal the current stored owner.
  - **Impact**: It gates the operation but is otherwise unused by the library.
- Current ownership state
  - **Control**: Initialization and ownership-management paths determine the stored owner.
  - **Constraints**: The stored owner must exactly match `msg.sender`.
  - **Impact**: It determines whether checkpoint execution is reached.
- Existing rate-limiter configuration and state
  - **Control**: Administrative configuration and usage paths establish the stored values.
  - **Constraints**: Declared field widths apply, while global-disable and directional-enabled flags are not consulted.
  - **Impact**: Overrides, defaults, prior usage, and `lastUpdated` determine each resulting state and event.
- `block.timestamp`
  - **Control**: Consensus determines this environmental value.
  - **Constraints**: Conversion to `uint32` truncates modulo 2^{32} rather than reverting.
  - **Impact**: The narrowed timestamp drives both directional decay calculations and replaces `lastUpdated`.
- `msg.value`
  - **Control**: A caller can attempt to attach native value.
  - **Constraints**: The nonpayable function rejects any nonzero value.
  - **Impact**: Successful execution transfers no native value.
- `tx.origin`
  - **Control**: It identifies the originating EOA.

    - **Constraints**: The resolved path never reads it.
    - **Impact**: It has no effect on authorization or checkpointing.
- Current execution context and linked RateLimiterLib deployment
    - **Control**: Proxy routing and deployment linking determine storage context, event emitter, target address, and bytecode.
    - **Constraints**: Correct routing, linking, and deployed library bytecode are deployment assumptions.
    - **Impact**: The `DELEGATECALL` updates the current context's namespaced storage and emits logs from that context.

Branches and code coverage

Intended branches

- Authorized execution: an owner call forwards the complete decoded array through the internal helper to the linked library.
    - ☐ Test coverage
- Empty array: an empty array performs no writes and emits no checkpoint events.
    - ☐ Test coverage
- Nonempty array: each entry causes one ordered state write and one `RateLimitCheckpointed` event.
    - ☐ Test coverage
- Per-EID override selected: an EID with `overrideDefaultConfig` uses its stored configuration.
    - ☐ Test coverage
- Default config selected: an EID without an override, including an unknown EID, uses the current default.
    - ☐ Test coverage
- Uninitialized timestamp: a zero `lastUpdated` preserves usage and stores the current narrowed timestamp.
    - ☐ Test coverage
- Same or non-forward timestamp: a nonzero `lastUpdated >= currentTimestamp` preserves usage and replaces the timestamp.
    - ☐ Test coverage
- Zero usage: elapsed-time decay preserves a direction with zero usage.
    - ☐ Test coverage
- Zero effective limit: a zero limit preserves usage even when global or directional flags indicate inactivity.

- ☐ Test coverage

- Zero effective window: a zero window preserves usage even when global or directional flags indicate inactivity.
  - ☐ Test coverage
- Elapsed time reaches the configured window: nonzero usage becomes zero when elapsed time reaches the window.
  - ☐ Test coverage
- Partial-window replenishment clears usage: usage becomes zero when floored replenishment meets or exceeds prior usage.
  - ☐ Test coverage
- Partial-window replenishment leaves usage: usage decreases by floored replenishment when replenishment is smaller.
  - ☐ Test coverage
- Elapsed-zero decay guard is unreachable on this path: the state helper returns before `_decayUsage` whenever elapsed time is zero.
  - ☐ Test coverage
- Shared timestamp: both directions use one `lastUpdated`, while independently floored recovery can discard fractional replenishment.
  - ☐ Test coverage
- Repeated EID: a duplicate repeats the write and event at the same timestamp without additional decay.
  - ☐ Test coverage
- Checkpoint before default-config replacement: pre-checkpointing materializes decay under the old default before replacement, preserving the timing semantics of the old configuration up to the checkpoint.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a failed `onlyOwner` check reverts before the library call, writes, or events.
  - ☐ Negative test
- Native-value attachment: any nonzero `msg.value` reverts before authorization or body execution.
  - ☐ Negative test
- Malformed calldata: invalid offsets, lengths, data size, or noncanonical `uint32` words prevent function entry.
  - ☐ Negative test

- No EID-policy rejection: zero, unknown, repeated, and unsorted EIDs are accepted and processed.
  - ☐ Negative test
- Library-call failure or gas exhaustion: any library failure or insufficient gas atomically rolls back all writes and logs.
  - ☐ Negative test
- Large owner-selected array: an array exceeding available gas reverts atomically because no explicit length cap exists.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The generated `this.` depicts a self-call, but the source performs an internal nonvirtual helper call followed by a fixed-target library `DELEGATECALL`. There is no value transfer, callback, further external call, or return value.
- **Manual call-path verification**: The verified path is `checkpointRateLimits` → internal `RateLimiterModule._checkpointRateLimits` → linked `RateLimiterLib.checkpointRateLimits` via `DELEGATECALL` → per-EID configuration selection, decay, storage write, and event; reverts bubble atomically.

Function: `constructor(address _token, address _lzEndpoint)`

This nonpayable implementation constructor executes `OFTAdapterUpgradeable(_token, _lzEndpoint)` and then calls `_disableInitializers()`. Deployment governance must select the intended token and endpoint because this derived layer has no deployer authentication or allowlist.

Inputs

- `_token`
  - **Control**: Deployment init code or factory logic supplies `_token`, which is forwarded unchanged to the base constructor.
  - **Constraints**: This derived constructor performs no explicit address, code-size, interface, decimals, or token-behavior validation.
  - **Impact**: The accepted value becomes `innerToken`, determining the SafeERC20 target used by later bridge debit and credit operations.
- `_lzEndpoint`

- **Control**: Deployment init code or factory logic supplies `_lzEndpoint`, which is forwarded unchanged to the base constructor.
- **Constraints**: This derived constructor performs no explicit address, code-size, interface, or endpoint-trust validation.
- **Impact**: The accepted value configures inherited OFT messaging, including later dispatch, delivery, and fee handling.

Branches and code coverage

Intended branches

- With zero creation value, base construction completes and the deployed implementation supports an initialized proxy.
  - ☑ Test coverage (`RateLimiterModuleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`, `RateLimiterModuleTest.test_QuoteOFTMax_FollowsDecimalConversionRate`)
- `_disableInitializers()` locks direct implementation initialization.
  - ☐ Test coverage
- Successful locking prevents direct initialization of the implementation while leaving proxy storage separately initializable through delegatecall.
  - ☐ Test coverage
- Base-accepted token and endpoint values remain the configuration used by later token transfers and inherited OFT operations.
  - ☑ Test coverage (`RateLimiterModuleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`, `RateLimiterModuleTest.test_QuoteOFTMax_FollowsDecimalConversionRate`)

Negative behavior

- Nonzero `msg.value` is rejected by the nonpayable constructor boundary, so no implementation is deployed.
  - ☐ Negative test
- A base-constructor revert stops execution before the derived body and rolls back all construction state and logs.
  - ☐ Negative test
- An `_disableInitializers()` revert rolls back preceding construction effects and prevents deployment.
  - ☐ Negative test
- This derived layer accepts deployment parameters that pass the base constructors

even when they are not authorized for the intended deployment.

- ☐ Negative test

Function call analysis

- `OFTAdapterUpgradeable(_token, _lzEndpoint)`
  - This statically selected base constructor receives both deployment-controlled addresses unchanged, runs before the derived body, and atomically aborts creation on failure. Dependency-defined validation, events, interactions, or callbacks cannot be enumerated from the supplied sources.
- `_disableInitializers()`
  - This unconditional internal helper runs after successful base construction in implementation storage, exposes no explicit external callback at this call site, and aborts deployment if it reverts.
- **Manual call-path verification**: ASI emitted no generated FCA sink header, but manual tracing recovered exactly `OFTAdapterUpgradeable(_token, _lzEndpoint)` followed by `_disableInitializers()`; the builder errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern separate Tempo resolution.

Function: `initialize(address _delegate)`

This function initializes proxy OFT delegation and ownership once. The implementation constructor disables its own initializers. Because callers are unrestricted, deployment must initialize the proxy atomically or prevent other callers from reaching it.

Inputs

- `_delegate`
  - **Control**: Selected by the first caller admitted by the inherited `initializer` life cycle.
  - **Constraints**: It is forwarded unchanged without source-local validation, while dependency code determines whether zero is accepted.
  - **Impact**: It becomes both the configured endpoint delegate and final ownership target.
- `msg.sender`
  - **Control**: The account or contract invoking `initialize` through the proxy.
  - **Constraints**: No role, ownership, signature, or equality check restricts the caller.

    - **Impact**: It is observed by inherited code, but ownership is transferred to `_delegate`, and the endpoint sees the proxy as caller.
- `msg.value`
    - **Control**: Selected by the caller when constructing the transaction.
    - **Constraints**: It must be zero because the function is nonpayable.
    - **Impact**: No ETH reaches the initializer or endpoint interaction.
- Proxy initialization state
    - **Control**: Determined by prior initialization activity in proxy storage.
    - **Constraints**: The inherited life cycle must admit the call, while the implementation is locked by `_disableInitializers()`.
    - **Impact**: It gates all setup, and any revert atomically rolls back initialization effects.
- Configured endpoint
    - **Control**: Fixed by `_lzEndpoint` during implementation construction rather than initializer calldata.
    - **Constraints**: This function provides no endpoint parameter or setter.
    - **Impact**: It receives `setDelegate(_delegate)` and may complete, revert, or attempt callback control flow.

Branches and code coverage

Intended branches

- Fresh proxy initialization assigns the supplied delegate as owner, enabling the owner-gated default configuration used by the test.
    - ☑ Test coverage (`RateLimiterModuleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- Initialization executes endpoint delegate setup and both ownership transitions in order.
    - ☐ Test coverage
- When `_delegate != msg.sender`, the caller remains implicit while `_delegate` independently becomes delegate and owner.
    - ☐ Test coverage
- When `_delegate == address(0)`, zero is forwarded unchanged and dependency behavior determines success.
    - ☐ Test coverage

Negative behavior

- Reinitialization after consuming the applicable initializer version must revert before setup repeats.

- ☐ Negative test

- Recursive external initialization while initialization is active must be rejected by the life-cycle guard.
  - ☐ Negative test
- Direct implementation initialization must fail because its constructor disables initializers.
  - ☐ Negative test
- A call carrying nonzero `msg.value` must fail before executing the function body.
  - ☐ Negative test
- An `endpoint.setDelegate(_delegate)` revert must propagate and roll back proxy initialization effects.
  - ☐ Negative test
- A later ownership-operation revert must also roll back the earlier endpoint state change.
  - ☐ Negative test

Function call analysis

- `this.__OFTCore_init(_delegate) -> this.__OAppCore_init(_delegate) -> this.__OAppCore_init_unchained(_delegate) -> this.endpoint.setDelegate(_delegate)`
  - The `_delegate` parameter is forwarded unchanged to the configured endpoint. No ETH or return data is used, reverts propagate, and callbacks occur before ownership initialization completes.
- **Manual call-path verification**: The `this.*` prefixes misrepresent inherited internal dispatch. The trace also omits the later internal `__Ownable_init()` and `_transferOwnership(_delegate)` calls. Imported dependency bodies determine endpoint validation and precise ownership checks and events.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This is an owner-only setter that replaces the complete default configuration, leaves per-EID state unchanged, and emits `DefaultRateLimitConfigSet` without `overrideDefaultConfig`. Later state materialization uses the replacement limit/window pairs even for disabled directions.

Inputs

- `_defaultConfig`

- **Control**: The caller supplies all configuration fields, while `onlyOwner` gates access and the linked-library target is fixed.
- **Constraints**: Enabled directions require nonzero limits and windows, disabled values are unrestricted, ABI widths apply, and `msg.value` must be zero.
- **Impact**: The struct replaces the stored default and affects later non-overridden EID resolution and decay without changing existing per-EID states.

Branches and code coverage

Intended branches

- An authorized valid update stores an effective default used by later adapter debit and credit accounting.
  - ☑ Test coverage (`RateLimiterModuleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- Success emits `DefaultRateLimitConfigSet`.
  - ☐ Test coverage
- When outbound is disabled, any outbound limit/window pair is accepted, stored, and emitted.
  - ☐ Test coverage
- When inbound is disabled, any inbound limit/window pair is accepted, stored, and emitted.
  - ☐ Test coverage
- Non-overridden tested EIDs resolve to the replacement default during later adapter debit and credit accounting.
  - ☑ Test coverage (`RateLimiterModuleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- Absence of checkpointing inside the setter is not independently instrumented.
  - ☐ Test coverage
- Later state materialization uses disabled-direction limit/window pairs, so nonzero values can decay usage while zero values preserve it.
  - ☐ Test coverage
- Replacement applies new decay slopes to uncheckpointed usage, while checkpointing affected EIDs first preserves old-policy accrued decay; the reproduction passed 1/1 with `1,000` uncheckpointed versus `910` checkpointed but is not exact wrapper coverage.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts before validation, storage writes, or event emission.
  - ☐ Negative test
- Enabled outbound with a zero limit reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Enabled outbound with a nonzero limit but zero window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Enabled inbound with a zero limit reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Enabled inbound with a nonzero limit but zero window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Invalid ABI calldata or nonzero `msg.value` reverts before the state-changing body.
  - ☐ Negative test
- Any linked-library failure propagates and atomically rolls back state and logs.
  - ☐ Negative test

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`
  - The wrapper calls the nonvirtual helper internally, then delegatecalls the fixed linked library in proxy storage context. Caller-controlled fields are validated, no return data is used, and the library makes no nested calls.
- **Manual call-path verification**: The source confirms direct internal dispatch to `RateLimiterModule._setDefaultRateLimitConfig` followed by the linked-library delegatecall, so ASI's `this.*` edge is inaccurate while its library continuation is correct.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This is an owner-only batch setter that checkpoints each EID under its prior effective configuration, stores replacements, and emits checkpoint/config events. Duplicate EIDs execute sequentially, while empty batches are no-ops.

Inputs

- `_params`

- **Control**: The owner selects each EID, override and enabled flags, windows, and limits, while `msg.sender`, prior/default state, `uint32(block.timestamp)`, and gas also affect execution.
- **Constraints**: Each enabled direction requires a nonzero limit and window, but there are no length, EID, uniqueness, ordering, or existing-usage bounds.
- **Impact**: Each element checkpoints floored decay under its prior effective config, writes state and config, and emits both events, with both directions sharing `lastUpdated`.

Branches and code coverage

Intended branches

- Authorized empty batches execute zero iterations and change no rate-limit storage or events.
  - ☐ Test coverage
- Enabled outbound configurations pass validation when both outbound limit and window are nonzero.
  - ☐ Test coverage
- Enabled inbound configurations pass validation when both inbound limit and window are nonzero.
  - ☐ Test coverage
- Disabled directions accept zero or nonzero inactive fields, which respectively preserve usage or permit decay.
  - ☐ Test coverage
- A prior configuration with `overrideDefaultConfig == true` supplies the effective checkpoint parameters.
  - ☐ Test coverage
- A prior configuration with `overrideDefaultConfig == false` causes checkpointing under the current default.
  - ☐ Test coverage
- When `lastUpdated == 0` or is not earlier than the cast timestamp, usage is retained and `lastUpdated` is set to the cast timestamp, which may remain unchanged or decrease.
  - ☐ Test coverage
- Partial elapsed windows reduce usage by floored `limit * elapsed / window`, bounded at zero.
  - ☐ Test coverage
- A full elapsed window clears nonzero usage when the applicable limit and window are nonzero.

- ☐ Test coverage

- Duplicate EIDs process sequentially, emit an event pair per occurrence, and leave the final config stored.
  - ☐ Test coverage
- Non-override replacements are stored and emitted, but future effective lookups continue using the default.
  - ☐ Test coverage

Negative behavior

- A non-owner caller reverts before the internal module or linked library executes.
  - ☐ Negative test
- Any nonzero `msg.value` is rejected because the entry point is nonpayable.
  - ☐ Negative test
- Malformed array, struct, integer, or boolean ABI encoding reverts during calldata decoding.
  - ☐ Negative test
- An enabled outbound direction with a zero limit or window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- An enabled inbound direction with a zero limit or window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Any later validation, arithmetic, gas, or delegatecall failure atomically rolls back earlier writes and logs.
  - ☐ Negative test
- Empty arrays, duplicate or zero EIDs, limits below usage, and disabled placeholders are otherwise accepted owner choices.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
  - The trace's first edge is an internal, nonvirtual module call. The only external control transfer is a link-fixed library delegatecall in proxy storage context, with no parameter-selected target, return value, further call, or reachable callback.
- **Manual call-path verification**: The verified path is

FraxOFTAdapterUpgradeable.setRateLimitConfigs → internal RateLimiterModule._setRateLimitConfigs → linked RateLimiterLib.setRateLimitConfigs → validation, effective-config checkpoint, state/event write, and config/event write. Failures bubble atomically, and separate default-transition behavior belongs to setDefaultRateLimitConfig.

Function: setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)

This is an owner-only, nonpayable setter that delegates to linked RateLimiterLib to store the global-disable flag in adapter or proxy namespaced storage and emit RateLimitGlobalConfigSet. It leaves default and per-EID configurations and usage states unchanged.

Inputs

- _globalConfig
  - **Control**: The owner supplies _globalConfig.isGloballyDisabled, which is forwarded unchanged to the linked library.
  - **Constraints**: Calldata must contain a canonical ABI boolean, msg.sender must be the owner, and msg.value must be zero.
  - **Impact**: The flag is written to slot 0xf02f50e1c246a23e457c592e128a403f9c1cc60baa3a1ad2dc69bb4a5a51be00, enabling or bypassing later rate-limit accounting without changing existing configurations or usage states.

The adapter or proxy address determines storage and log context, while deployment-time linking selects the trusted library target. This path does not read tx.origin, timestamps, token state, endpoint state, fees, or virtual hooks.

Branches and code coverage

Intended branches

- Globally disable rate limits: an authorized true stores the flag and emits the event, causing later accounting to be bypassed and availability to report unlimited while quoteOFT retains its uint64 shared-decimal cap.
  - ☐ Test coverage
- Globally enable rate limits: an authorized false stores the flag and emits the event, resuming enforcement against existing configurations and time-decayed usage state.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller that fails `onlyOwner` reverts before the helper, delegatecall, storage write, or event.
  - ☐ Negative test
- Nonzero native value: a call with nonzero `msg.value` reverts at the nonpayable entry point without persistent effects.
  - ☐ Negative test
- Malformed calldata: invalid struct encoding or a noncanonical ABI boolean reverts during decoding before the function body executes.
  - ☐ Negative test
- Library revert or delegatecall failure: any linked-library revert or failed delegatecall atomically rolls back the storage write and event.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The owner controls only the boolean. The actual path is an internal call to nonvirtual `_setRateLimitGlobalConfig`, followed by a fixed linked-library `DELEGATECALL` that writes and emits in adapter or proxy context. The delegatecall has no semantic return value or reentrancy edge and reverts atomically on failure.
- **Manual call-path verification**: The ASI `this.` prefix is incorrect because the helper call is internal, while the subsequent linked `RateLimiterLib.setRateLimitGlobalConfig` delegatecall is correct.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

Owner-only administrative reseeding overwrites namespaced per-EID usage and shared timestamps through the linked library without consuming capacity, changing configuration, or checkpointing decay. Entries execute in order; duplicates emit individually, and the last write wins.

Inputs

- `_params`
  - **Control**: The owner controls the ordered array and all decoded fields, while `block.timestamp` supplies validation time, `msg.value` must be zero, and gas bounds batch size.
  - **Constraints**: ABI widths are enforced, each timestamp must not exceed `block.timestamp`, and a zero timestamp requires both usages to be zero,

while empty arrays and duplicate EIDs are allowed.

- **Impact**: Each valid entry overwrites `states[eid]` and emits `RateLimitStateSet`, with usages accepted independently of configured limits and the final duplicate determining stored state.

Branches and code coverage

Intended branches

- Authorized dispatch: with the owner as `msg.sender` and zero value, the wrapper calls the internal helper once, which invokes the linked library once.
  - ☐ Test coverage
- Empty array: an empty array succeeds without validation, storage writes, or events.
  - ☐ Test coverage
- Valid nonempty array: each valid entry overwrites `states[eid]` and emits `RateLimitStateSet` before processing the next entry.
  - ☐ Test coverage
- Duplicate EIDs: duplicate EIDs execute sequentially, emit once per entry, and leave the last supplied state stored.
  - ☐ Test coverage
- State independent of active limits: valid `uint112` usages are accepted even when they exceed effective limits because configuration is not compared.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a non-owner call reverts before the helper executes, producing no writes or events.
  - ☐ Negative test
- Native value supplied: nonzero `msg.value` reverts at nonpayable function entry before authorization or processing.
  - ☐ Negative test
- Malformed or out-of-range ABI data: invalid array encoding or values outside declared ABI widths revert before the body executes.
  - ☐ Negative test
- Future timestamp: any `lastUpdated` greater than `block.timestamp` reverts with `InvalidRateLimitState` before that entry is written or emitted.
  - ☐ Negative test
- Zero timestamp with nonzero usage: a zero timestamp with either usage nonzero reverts with `InvalidRateLimitState` before that entry is written or emitted.

- ☐ Negative test

- Later-element failure or out-of-gas: any later validation failure, library revert, or out-of-gas condition atomically rolls back all preceding writes and logs.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
  - The trace's `this.` prefix is incorrect. The source makes an unqualified internal call to nonvirtual `_setRateLimitStates`, followed by a statically linked library `DELEGATECALL` that preserves proxy storage context, original sender, and zero value. The `_params` argument cannot select the target, and the library makes no further calls or callbacks.
- **Manual call-path verification**: The verified path is `setRateLimitStates` → internal `RateLimiterModule._setRateLimitStates` → linked `RateLimiterLib.setRateLimitStates` by `DELEGATECALL`, with reverts bubbling atomically and no application-level return value.

5.6. Module: FraxOFTMintableAdapterUpgradeable.sol

Function: `checkpointRateLimits(uint32[] _eids)`

Owner-only checkpointing applies each EID's override or default configuration, updates both usages and their shared timestamp, and emits an event. Directional and global enable flags are ignored.

Inputs

- `_eids`
  - **Control**: The owner selects the `uint32` EIDs, array length, order, and duplicates, while the timestamp, configurations, and states are implicit inputs.
  - **Constraints**: Calldata must decode canonically, `msg.value` must be zero, and empty, duplicate, zero, unconfigured, or disabled EIDs are accepted.
  - **Impact**: Each entry selects its state and effective configuration, writes both usages and `lastUpdated`, and emits `RateLimitCheckpointed`, with any failure reverting the entire batch.

Branches and code coverage

Intended branches

- An empty array reaches the library but performs no iterations, writes, or checkpoint events.
  - ☐ Test coverage
- A nonempty array processes every entry in order and emits one checkpoint event per successful position.
  - ☐ Test coverage
- When `overrideDefaultConfig` is true, the EID's stored override supplies both directions' limit and window fields.
  - ☐ Test coverage
- When `overrideDefaultConfig` is false, `defaultConfig` applies; checkpointing before its replacement uses the old configuration's slope and advances time.
  - ☐ Test coverage
- When `lastUpdated == 0`, usage is preserved while the current truncated timestamp is stored and emitted.
  - ☐ Test coverage
- When `lastUpdated >= uint32(block.timestamp)`, usage is preserved and the shared timestamp is set to the current truncated value.
  - ☐ Test coverage
- When elapsed time is below the window, usage decays by floored proportional recovery, and both directions share elapsed time and `lastUpdated`.
  - ☐ Test coverage
- When elapsed time reaches or exceeds a nonzero window and the limit is nonzero, the corresponding usage decays to zero.
  - ☐ Test coverage
- Zero usage, limit, or window preserves that direction's usage while checkpointing can still advance the shared timestamp.
  - ☐ Test coverage
- Disabled directional or global flags do not gate checkpointing, so stored nonzero fields can still decay usage.
  - ☐ Test coverage
- A duplicate EID has no additional elapsed time but rewrites the state and emits another event.
  - ☐ Test coverage
- Distinct EIDs use independent mapping entries, with all writes and events committing

atomically.

- ☐ Test coverage

Negative behavior

- A caller other than `owner()` reverts in `onlyOwner` before the body executes.
  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable entry point.
  - ☐ Negative test
- Malformed calldata or noncanonical `uint32` elements revert during ABI decoding before mutation.
  - ☐ Negative test
- A delegatecall failure, panic, or out-of-gas condition bubbles up and rolls back all prior writes and events.
  - ☐ Negative test
- A batch exceeding available gas fails atomically without preserving a processed prefix.
  - ☐ Negative test
- Empty, duplicate, zero, unconfigured, directionally disabled, or globally disabled EIDs are intentionally not rejected.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The source path is an internal `RateLimiterModule._checkpointRateLimits` call followed by deployment-linked RateLimiterLib execution via `DELEGATECALL` in proxy context. The owner controls only `_eids`, the library makes no further external calls, and failures revert atomically.
- **Manual call-path verification**: The path is `checkpointRateLimits` → `RateLimiterModule._checkpointRateLimits` → linked `RateLimiterLib.checkpointRateLimits` → effective-config selection and usage decay → state write and event. ASI misrenders the internal edge as `this._checkpointRateLimits`, but the linked-library `DELEGATECALL` is real.

Function: `constructor(address _token, address _lzEndpoint)`

Construction forwards `_token` and `_lzEndpoint` unchanged to the base constructor and then calls `_disableInitializers()`. This locks implementation storage only. Proxy initialization and inherited behavior cannot be established from the supplied source.

Inputs

- `_token`
  - **Control**: The deploying EOA or factory controls the address without target-local authorization.
  - **Constraints**: The value must ABI-decode as an address and is forwarded without local validation.
  - **Impact**: Base code determines validation and persistence, while an incorrect token may break later adapter behavior.
- `_lzEndpoint`
  - **Control**: The deploying EOA or factory controls the address without a registry or canonical-endpoint check.
  - **Constraints**: The value must ABI-decode as an address and is forwarded unchanged to the base constructor.
  - **Impact**: Base code determines endpoint configuration, while an incorrect endpoint may invalidate inherited messaging behavior.

Branches and code coverage

Intended branches

- A zero-value deployment with valid token and endpoint arguments installs runtime code exposing the expected token and version.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Token_returnsExpected`, `FraxOFTMintableAdapterUpgradeableTIP20Test.test_Version_returnsExpected`)
- Exact ordering of base construction before the initializer lock is not instrumented.
  - ☐ Test coverage
- Unconditional implementation lock: every successful execution of the target constructor body invokes `_disableInitializers()`.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Implementation_CannotE`
- No scoped virtual-hook dispatch: construction does not call runtime hooks, rate-limit helpers, supply-tracking helpers, or linked libraries.
  - ☐ Test coverage

Negative behavior

- Nonzero creation value: because the constructor is nonpayable, creation with `msg.value > 0` reverts.

- ☐ Negative test

- Malformed constructor arguments: creation fails when calldata cannot ABI-decode as two addresses.
  - ☐ Negative test
- Base-constructor failure: any base-constructor revert, panic, decode failure, or transitive-call failure atomically aborts creation.
  - ☐ Negative test
- Initializer-lock failure: a revert from `_disableInitializers()` aborts deployment and rolls back prior construction effects.
  - ☐ Negative test
- Locally unchecked configuration: the target does not locally reject zero, EOA, incompatible, or unintended token and endpoint addresses.
  - ☐ Negative test

Function call analysis

- `OFTAdapterUpgradeable(_token, _lzEndpoint)`
  - This base-constructor invocation receives both deployer-controlled addresses unchanged, executes before the derived body, and may contain unavailable validation, state changes, events, or transitive external calls whose failures revert creation.
- `_disableInitializers()`
  - This inherited internal helper executes after successful base construction, locks only implementation storage, and aborts deployment if it reverts.
- EVM contract-creation boundary
  - An EOA or factory supplies creation code and arguments, with any `CREATE2` salt factory-controlled. Success installs runtime bytecode, while failure atomically rolls back construction and transitive effects.
- **Manual call-path verification**: ASI emitted no FCA sink header. The verified path is `OFTAdapterUpgradeable(_token, _lzEndpoint)` followed by `_disableInitializers()`. It has no runtime hook or linked-library dispatch. Tempo sibling `_payNative` errors do not affect this constructor.

Findings

- Safety depends on the intended token, endpoint, inherited bytecode, token roles, and proxy initialization and upgrade procedure.

Function: `recover()`

The permissionless `recover()` queries the configured token for the balance of the executing adapter or proxy. It returns if the reported balance is zero; otherwise, it transfers that exact balance to the current owner. It writes no adapter state and ignores the decoded transfer boolean.

Inputs

- The function takes no explicit parameters. Any caller controls timing only, and nonzero `msg.value` is rejected before execution.
- Implicit inputs are deployment-fixed `innerToken`, `address(this)`, the current `owner()` (which may be zero), and the token-reported balance. Correct proxy token configuration is a deployment assumption.

Branches and code coverage

Intended branches

- If `balanceOf(address(this))` reports zero, the function returns before reading `owner()` or attempting a transfer.
  - ☐ Test coverage
- If the reported balance is nonzero, the function forwards that exact `uint256` to `innerToken.transfer(owner(), balance)`.
  - ☐ Test coverage
- If `transfer` returns an ABI-valid `true`, the ignored result permits normal completion.
  - ☐ Test coverage
- If `transfer` returns an ABI-valid `false`, the ignored result still permits normal completion without proving token movement.
  - ☐ Test coverage
- No authorization branch exists, although callers cannot select the token, queried account, recipient, or amount.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` reverts at the nonpayable entry check before the function body executes.
  - ☐ Negative test
- A reverting, gas-exhausting, or malformed `balanceOf` response propagates before owner lookup or transfer.
  - ☐ Negative test

- If the reported balance exceeds the transferable amount, the token may revert, return malformed data, or return `false`, which `recover()` ignores.
  - ☐ Negative test
- A reverting, gas-exhausting, or malformed `transfer` response reverts the call subtree, although an upstream contract may catch the failure.
  - ☐ Negative test
- A zero current owner is passed directly to `transfer`, leaving zero-address handling to the configured token.
  - ☐ Negative test
- The unguarded transfer may reenter an externally reachable route, recurse, exhaust gas, or fail. The `recover()` function has no local partial-state invariant.
  - ☐ Negative test

Function call analysis

- `this.innerToken.balanceOf(address(this))`
  - The deployment-fixed token controls the reported balance and may revert, exhaust gas, or attempt callbacks. The decoded balance is checked for zero and otherwise forwarded unchanged. Static-call semantics prevent callbacks from changing state.
- `this.innerToken.transfer(this.owner(), balance)`
  - The source calls the internal `owner()` function, not external `this.owner()`. The state-capable token call receives the current owner and exact balance, may reenter, and may return an ABI-valid `true` or `false`; either result is ignored.
- **Manual call-path verification**: The source contains exactly two token interactions, and `transfer` occurs only when the balance is nonzero. ASI's `this.innerToken` notation does not indicate a self-call. Builder failures in sibling files do not affect this function's interactions.

Function: `setAllowNegativeSupply(uint32 _eid, bool _allow)`

This owner-only, nonpayable setter forwards both arguments unchanged, updates `allowNegativeSupply[_eid]`, and emits `SetAllowNegativeSupply`. It does not validate counters or move tokens or value. Proxy ownership and initialization are determined at deployment.

Inputs

- `_eid`

    - **Control**: The authorized owner selects the ABI-valid `uint32`, which is forwarded unchanged.
    - **Constraints**: No zero, peer, endpoint, initialization, or ledger validation is performed.
    - **Impact**: It directly selects the mapping key and indexed event argument governing later matching credit accounting.
- `_allow`
    - **Control**: The authorized owner selects the canonical boolean, which is stored and emitted unchanged.
    - **Constraints**: The setter accepts repeated values and does not validate any supply counter.
    - **Impact**: Setting it to `true` skips future transfer-from enforcement, and setting it to `false` enables that enforcement. Neither changes counters, `_debit`, or checked `totalSupply`.

Branches and code coverage

Intended branches

- The inherited owner can write the selected mapping entry and emit `SetAllowNegativeSupply`.
    - ☐ Test coverage
- Setting `_allow == true` skips future `_addToTotalTransferFrom` enforcement without repairing counters or `totalSupply`.
    - ☐ Test coverage
- Setting `_allow == false` enables the nonnegative-ledger check for later matching transfer-from accounting.
    - ☐ Test coverage
- The verified 1/1 probe shows that setting `_allow` to `false` when the ledger is already negative succeeds and blocks later accounting. The ledger continues to underflow until the counters are restored, and setting `_allow` back to `true` does not repair `totalSupply`.
    - ☐ Test coverage — audit-only probe — no scoped-repository test case was mapped
- Either boolean may be written repeatedly for any ABI-valid EID, including an uninitialized key.
    - ☐ Test coverage

Negative behavior

- A non-owner call reverts before the mapping write or event.

- ☐ Negative test

- Nonzero `msg.value` reverts because the function is nonpayable.
  - ☐ Negative test
- A noncanonical boolean or otherwise malformed ABI payload reverts during decoding.
  - ☐ Negative test
- Out-of-gas during authorization, storage access, `SSTORE`, or logging atomically rolls back storage and logs.
  - ☐ Negative test

Function call analysis

- `this._setAllowNegativeSupply(_eid, _allow) -> this._getSupplyTrackingStorage()`
  - The inherited calls are internal. The Yul helper only assigns the fixed module storage slot. The function then writes the mapping and emits the event without callbacks or reentrancy.
- **Manual call-path verification**: The path is `onlyOwner -> _setAllowNegativeSupply -> _getSupplyTrackingStorage -> SSTORE -> LOG`. ASI renders internal calls as `this.*` and omits the event. Later, `_credit` consumes limits, performs accounting, and calls the fixed `innerToken.minter_mint`. Any failure rolls back atomically. Failures in the sibling Tempo/TIP-20 `_quote` builder do not affect this path.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This owner-only setter forwards `_defaultConfig` unchanged through the internal module to a linker-fixed library delegatecall. It validates and replaces active-context defaults without checkpointing per-EID state, so existing default-backed usage is reinterpreted under the replacement decay slope.

Inputs

- `_defaultConfig`
  - **Control**: The owner supplies all three booleans, two `uint32` windows, and two `uint112` limits.
  - **Constraints**: Each enabled direction requires a nonzero limit and window, while `overrideDefaultConfig` does not select default fallback.
  - **Impact**: The complete struct is stored, six fields are emitted, and existing default-backed usage is reinterpreted under the replacement decay slope without a checkpoint.
- `msg.sender`

    - **Control**: The immediate caller supplies `msg.sender`.
    - **Constraints**: Inherited `onlyOwner` must authorize it before the setter executes.
    - **Impact**: It controls authorization but is neither stored nor emitted.
- `msg.value`
    - **Control**: The caller selects the attached native value.
    - **Constraints**: The nonpayable entry point requires `msg.value == 0`.
    - **Impact**: No native value is accepted, forwarded, or refunded.
- `address(this)` / execution context
    - **Control**: Proxy or direct invocation determines the active execution context.
    - **Constraints**: The linked library delegatecall preserves that context.
    - **Impact**: The context determines the storage written and event-emitting address.
- `block.timestamp`
    - **Control**: The block environment supplies the timestamp.
    - **Constraints**: This setter does not read or validate it.
    - **Impact**: No checkpoint is recorded, so later default-backed calculations apply replacement parameters across elapsed time.
- `tx.origin`
    - **Control**: The originating account determines `tx.origin`.
    - **Constraints**: This path never reads or compares it.
    - **Impact**: It has no authorization, storage, event, arithmetic, or interaction effect.

Branches and code coverage

Intended branches

- Successful configuration replacement: an authorized zero-value call with valid enabled pairs stores the complete struct and emits its six event fields.
    - ☐ Test coverage
- Outbound validation branch: disabled outbound values are unrestricted, while an enabled outbound direction requires a nonzero limit and window.
    - ☐ Test coverage
- Inbound validation branch: disabled inbound values are unrestricted, while an enabled inbound direction requires a nonzero limit and window.
    - ☐ Test coverage
- Default-backed EID transition without checkpointing: no checkpoint occurs, and the passing 1/1 reproduction exposed uncheckpointed usage of `1,000` versus checkpointed

usage of 910.

- ☐ Test coverage

- Per-EID override isolation: EIDs whose stored configuration has `overrideDefaultConfig == true` continue using that override.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a failed `onlyOwner` check prevents the delegatecall, write, and event.
  - ☐ Negative test
- Invalid enabled outbound configuration: a zero outbound limit or window while enabled reverts with `InvalidRateLimitConfig()` before storage or emission.
  - ☐ Negative test
- Invalid enabled inbound configuration: a zero inbound limit or window while enabled reverts with `InvalidRateLimitConfig()` before storage or emission.
  - ☐ Negative test
- Attached native value: nonzero `msg.value` is rejected before successful execution.
  - ☐ Negative test
- Library execution failure: any delegatecall failure propagates atomically, rolling back writes and events.
  - ☐ Negative test

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`
  - The first leg is an inherited internal call. The linker-fixed delegatecall preserves caller and context, has no return or callback, and propagates failures atomically.
- **Manual call-path verification**: The resolved path is `setDefaultRateLimitConfig` → `RateLimiterModule._setDefaultRateLimitConfig` → `RateLimiterLib.setDefaultRateLimitConfig` → validation → storage replacement → event. ASI's `this.*` rendering is incorrect, and sibling builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` do not affect this path.

Function: `setInitialTotalSupply(uint32 _eid, uint256 _amount)`

This owner-only setter overwrites namespaced `initialTotalSupply[_eid]` with `_amount` and emits `SetInitialTotalSupply`. It performs no value or consistency validation and affects later supply telemetry and inbound-credit limits, not token balances directly.

Inputs

- `_eid`
  - **Control**: Any caller may submit `_eid`, but only the owner, as determined from `msg.sender`, can reach the write.
  - **Constraints**: Every canonical `uint32` value is accepted without EID validation.
  - **Impact**: The `_eid` parameter selects the overwritten mapping entry, emitted indexed field, and later per-EID accounting.
- `_amount`
  - **Control**: The authorized owner supplies `_amount`. Value, balances, and external data do not affect it.
  - **Constraints**: Every `uint256` value is accepted without consistency checks, and repeated calls may replace it.
  - **Impact**: The `_amount` parameter becomes the stored and emitted value and may later cause checked arithmetic reverts or credit rejection.

Implicit inputs are calldata validity, `msg.sender`, zero `msg.value`, deployed ownership state, available gas, and the overwritten mapping value. The `tx.origin` value and external data are unused.

Branches and code coverage

Intended branches

- An authorized call writes the selected `initialTotalSupply[_eid]`, which is reflected in asserted storage and headroom.
  - ☑ Test coverage (`V12OSupplyGuardForkTest.test_UpgradeThenSeed_RestoresIntendedHeadroom`, `V12OSupplyGuardForkTest.test_UpgradeThenSeed_SurplusEid_HeadroomEquals`, `FraxOFTMintableAdapterUpgradeableTIP2OTest.test_SetInitialTotalSupply_succeeds`)
- If an entry already exists, the authorized call replaces rather than accumulates it.
  - ☐ Test coverage
- Boundary values for both arguments are accepted subject only to ordinary EVM execution.

- ☐ Test coverage

Negative behavior

- An unauthorized caller reverts.
  - ☑ Negative test (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_SetInitialTotalSupply_`
- Unchanged storage and event state are not independently asserted.
  - ☐ Negative test
- Malformed or noncanonical calldata reverts during ABI decoding before execution.
  - ☐ Negative test
- Nonzero `msg.value` triggers the nonpayable check and reverts without effects.
  - ☐ Negative test
- Out-of-gas or another EVM failure propagates and atomically rolls back storage and logs.
  - ☐ Negative test
- An inconsistent owner-selected amount may later make supply telemetry revert or inbound-credit accounting reject credits.
  - ☐ Negative test

Function call analysis

- `this._setInitialTotalSupply(_eid, _amount) -> this._getSupplyTrackingStorage()`
  - Despite the ASI-rendered `this.*` notation, this trace represents an internal, nonvirtual call. The private accessor only assigns fixed slot `0x419276e85a544278a01dfd89b03028910afb9d04e0edf9a7b0d319d61e5bb200`. There is no external call, return data, callback, or reentrancy point.
- **Manual call-path verification**: The verified path is `setInitialTotalSupply` → internal `_setInitialTotalSupply` → private `_getSupplyTrackingStorage` → overwrite mapping → emit event. The generated `this.*` calls are incorrect. Builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol:174` and `FraxOFTUpgradeableTempo.sol:48` concern sibling `_quote` expressions.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This owner-only batch function checkpoints the old effective configurations before storing validated replacements. It does not read the global-disable flag or return a value.

Inputs

- `_params`
  - **Control**: The owner controls array length, order, repeated EIDs, and every configuration field.
  - **Constraints**: Enabled directions require nonzero limits and windows, while disabled-direction fields are unrestricted.
  - **Impact**: Each element checkpoints under the old effective configuration, stores the exact proposal, and emits both events.
- `msg.sender`
  - **Control**: The immediate caller selects `msg.sender`, which is preserved through the library delegatecall.
  - **Constraints**: It must satisfy `onlyOwner`.
  - **Impact**: It controls authorization but is otherwise unused and unlogged.
- `msg.value`
  - **Control**: The caller selects the transaction value.
  - **Constraints**: It must be zero because the function is nonpayable.
  - **Impact**: Nonzero value prevents execution, and no ETH is transferred.
- `tx.origin`
  - **Control**: The transaction originator determines `tx.origin`.
  - **Constraints**: No constraint is applied in this path.
  - **Impact**: It is not read and has no effect.
- `block.timestamp`
  - **Control**: The block environment supplies the timestamp.
  - **Constraints**: It is narrowed to `uint32`, assuming migration before the 2106 boundary.
  - **Impact**: It determines elapsed decay and the checkpointed `lastUpdated`.

Branches and code coverage

Intended branches

- Authorized empty batch: an owner-supplied empty array succeeds without storage changes or events.
  - ☐ Test coverage
- Authorized nonempty batch: each element is validated, checkpointed, stored, and emitted in order.
  - ☐ Test coverage
- Outbound validation passes: outbound is disabled or has both a nonzero limit and

window.

- ☐ Test coverage

- Inbound validation passes: inbound is disabled or has both a nonzero limit and window.
  - ☐ Test coverage
- Old effective config uses an override: an existing enabled override supplies the old checkpoint parameters.
  - ☐ Test coverage
- Old effective config uses the default: without an existing override, `defaultConfig` supplies the old checkpoint parameters.
  - ☐ Test coverage
- Uninitialized/equal/future checkpoint timestamp: usage is preserved when `lastUpdated` is zero or not earlier than the narrowed timestamp.
  - ☐ Test coverage
- Elapsed-time checkpoint: positive elapsed time decays both usages under the old effective configuration.
  - ☐ Test coverage
- No-decay parameter branch: usage is preserved when usage, the old limit, or the old window is zero, so owner-selected inactive fields control disable/reenable history.
  - ☐ Test coverage
- Full-window decay branch: nonzero usage becomes zero when the elapsed time reaches the old window.
  - ☐ Test coverage
- Partial-window decay branch: usage is reduced by capped proportional replenishment when the elapsed time is below the old window.
  - ☐ Test coverage
- Repeated EID with override enabled: a later occurrence observes the earlier stored override, while the shared timestamp preserves its checkpointed state.
  - ☐ Test coverage
- Repeated EID with override disabled: a later occurrence checkpoints against `defaultConfig`, and the final occurrence determines stored configuration.
  - ☐ Test coverage
- Atomic success: all writes and paired events commit only after every element succeeds.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a non-owner reverts before the helper or library call.

- ☐ Negative test
- ETH supplied: nonzero `msg.value` is rejected before body execution.
  - ☐ Negative test
- Malformed ABI input: invalid offsets, lengths, booleans, or narrow integers fail decoding.
  - ☐ Negative test
- Invalid outbound config: enabled outbound with a zero limit or window reverts before that element is checkpointed.
  - ☐ Negative test
- Invalid inbound config: enabled inbound with a zero limit or window reverts before that element is checkpointed.
  - ☐ Negative test
- Later-element failure: any later revert, panic, out-of-gas condition, or failed library call rolls back all earlier writes and logs.
  - ☐ Negative test
- Excessive batch gas use: an uncapped batch exceeding available gas reverts atomically.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
  - The source calls the internal nonvirtual `_setRateLimitConfigs`, which delegates to the statically linked library through `DELEGATECALL`. This preserves the sender and the proxy or implementation storage context without further calls or callbacks.
- **Manual call-path verification**: The exact path is `setRateLimitConfigs` → `RateLimiterModule._setRateLimitConfigs` → linked `RateLimiterLib.setRateLimitConfigs`. The generated `this._setRateLimitConfigs` receiver is incorrect, and deployment trusts the reviewed linked bytecode.

Function: `setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)`

This owner-only setter delegates to the deployment-linked library, overwrites namespaced `globalConfig`, and emits `RateLimitGlobalConfigSet` from the caller/proxy. Enabling the flag bypasses accounting, while disabling it resumes enforcement without changing configurations, usage, timestamps, or ownership.

Inputs

- `_globalConfig`
  - **Control**: The authorized caller supplies the sole boolean member unchanged.
  - **Constraints**: Calldata must contain a canonical boolean, and either value is accepted.
  - **Impact**: The value replaces namespaced `globalConfig` and controls subsequent global rate-limit enforcement.
- `msg.sender` and inherited owner state
  - **Control**: The immediate caller supplies `msg.sender`, while the active execution context stores the owner.
  - **Constraints**: Inherited `onlyOwner` requires `msg.sender == owner()`.
  - **Impact**: These values gate access without changing ownership or the stored configuration payload.
- `msg.value`
  - **Control**: The caller chooses whether to attach native value.
  - **Constraints**: The nonpayable entry point requires `msg.value == 0`.
  - **Impact**: No native value reaches the internal setter or library.
- Deployment-linked library address
  - **Control**: Deployment-time linking fixes the library address independently of calldata.
  - **Constraints**: Execution requires the linked library call to succeed against trusted bytecode.
  - **Impact**: The library executes through `DELEGATECALL` against caller/proxy storage and emits from that address.

Branches and code coverage

Intended branches

- An authorized zero-value call with `isGloballyDisabled == true` stores `true`, emits `RateLimitGlobalConfigSet(true)`, and enables the global bypass.
  - ☐ Test coverage
- An authorized zero-value call with `isGloballyDisabled == false` stores `false`, emits `RateLimitGlobalConfigSet(false)`, and resumes enforcement without clearing existing state.
  - ☐ Test coverage

Negative behavior

- A caller other than the owner reverts before the internal setter or linked library executes.
  - ☐ Negative test
- Malformed struct calldata, including a noncanonical boolean, fails during ABI decoding.
  - ☐ Negative test
- Any nonzero `msg.value` is rejected before function-body execution.
  - ☐ Negative test
- A linked-library failure propagates and atomically removes all writes and logs.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The `this.` edge is source-inaccurate because the wrapper makes an unqualified internal call, but the subsequent fixed linked-library `DELEGATECALL` is real and has no return value, callback, or further external edge.
- **Manual call-path verification**: The source resolves the path as `FraxOFTMintableAdapterUpgradeable.setRateLimitGlobalConfig` → internal nonvirtual `RateLimiterModule._setRateLimitGlobalConfig` → linked `RateLimiterLib.setRateLimitGlobalConfig` → namespaced assignment and event. Library failure rolls back atomically, and sibling-file parser failures do not alter this path.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

This owner-only wrapper replaces raw per-EID rate-limit states through the linked library and emits `RateLimitStateSet` per element. It does not change configuration, supply tracking, tokens, or LayerZero messages, and it returns no data.

Inputs

- `_params`
  - **Control**: The owner controls the ordered states, including duplicate or zero EIDs and empty batches, while the library reads `block.timestamp`.
  - **Constraints**: Zero timestamps require zero usages, nonzero timestamps cannot be future values, and usages need not respect effective limits.
  - **Impact**: Each element replaces the complete namespaced state and emits an event, with duplicate EIDs processed sequentially and the final value

retained.

Branches and code coverage

Intended branches

- Authorized empty batch: an authorized empty batch succeeds without writes or events.
  - ☐ Test coverage
- Valid zero state: a zero timestamp with both usages zero stores the all-zero state and emits one event.
  - ☐ Test coverage
- Valid timestamped state: a nonzero timestamp not later than `block.timestamp` stores and emits the supplied state regardless of effective limits.
  - ☐ Test coverage
- Ordered batch and duplicate EIDs: valid elements are processed in order. Each duplicate EID emits an event, and the final value remains stored.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller failing `onlyOwner` reverts before any library execution, write, or event.
  - ☐ Negative test
- Nonzero native value: the nonpayable entry point rejects nonzero `msg.value` before executing its body.
  - ☐ Negative test
- Future timestamp: any timestamp greater than `block.timestamp` causes `InvalidRateLimitState` and reverts the batch.
  - ☐ Negative test
- Nonzero usage with zero timestamp: a zero timestamp paired with either nonzero usage causes `InvalidRateLimitState`.
  - ☐ Negative test
- Later invalid element or execution failure: any later validation or execution failure atomically rolls back all preceding writes and logs.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
    - The generated trace suggests a self-call through `this`, but the source invokes internal `_setRateLimitStates`. It makes the fixed linked-library `DELEGATECALL` with owner-controlled arguments, preserves the caller and zero value, and uses the proxy's storage and event context. It returns no data and makes no nested callback-capable call.
- **Manual call-path verification**: The verified path is external wrapper → internal `RateLimiterModule._setRateLimitStates` → linked `RateLimiterLib.setRateLimitStates`. Reverts roll back the full batch. Unrelated ASI `_quote` parser failures do not indicate any omitted interaction in this path.

5.7. Module: FraxOFTMintableAdapterUpgradeableTIP20.sol

Function: `checkpointRateLimits(uint32[] _eids)`

This owner-only, nonpayable function checkpoints both usage directions for each EID through the linked library, updating caller storage and emitting events. It returns no value and performs no token or native-value transfer.

Inputs

- `_eids`
    - **Control**: The owner controls the array's length, order, values, emptiness, and duplicates, while `msg.sender` enforces ownership and `tx.origin` is unused.
    - **Constraints**: ABI decoding, zero `msg.value`, ownership, transaction gas, and the documented pre-`uint32` timestamp-rollover assumption apply, but no semantic EID validation occurs.
    - **Impact**: Each EID selects its override or default configuration and state, checkpoints both directions, and emits an event; empty arrays do nothing, and duplicates repeat processing.

Branches and code coverage

Intended branches

- Empty array: a zero-length array performs no writes and emits no checkpoint events.
    - ☐ Test coverage
- Per-EID override selection: an EID with `overrideDefaultConfig` set uses its stored directional limits and windows.

☐ Test coverage

- Default selection: an EID without `overrideDefaultConfig` set uses the default configuration; checkpointing before replacement materializes old-parameter decay, with the 1/1 reproduction yielding `910` checkpointed versus `1,000` uncheckpointed.

☐ Test coverage

- Uninitialized or nonadvancing timestamp: a zero or nonadvancing `lastUpdated` preserves usage and writes the current timestamp, including same-transaction duplicates.

☐ Test coverage

- Advancing timestamp: a later timestamp decays both directions over the elapsed interval and advances their shared timestamp.

☐ Test coverage

- No-decay guard: zero usage, limit, window, or elapsed time preserves usage while checkpointing can still advance the timestamp.

☐ Test coverage

- Full-window decay: when elapsed time reaches or exceeds the selected window, that direction's usage becomes zero.

☐ Test coverage

- Partial-window decay: when elapsed time is less than the selected window, usage decreases by `floor(limit * elapsed / window)`, bounded at zero, as demonstrated by the `1,000` to `910` reproduction.

☐ Test coverage

- Shared timestamp and rounding: both directions share one timestamp, and integer division discards fractional recovery.

☐ Test coverage

- Repeated EID: every occurrence rewrites and emits, but later same-timestamp occurrences do not decay usage again.

☐ Test coverage

Negative behavior

- Unauthorized caller: the `onlyOwner` modifier reverts before checkpoint execution, leaving no surviving writes or logs.

☐ Negative test

- Native value supplied: nonzero `msg.value` is rejected by nonpayable dispatch before the body executes.

☐ Negative test

- Malformed calldata: invalid `uint32[]` encoding reverts during ABI decoding before mutation.

- ☐ Negative test

- No semantic EID rejection: zero, unconfigured, repeated, or otherwise owner-selected `uint32` EIDs are processed rather than rejected.
  - ☐ Negative test
- Library revert or gas exhaustion: runtime failure, delegatecall failure, or gas exhaustion reverts all writes and logs atomically.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The trace's self-call does not occur. Direct internal dispatch reaches a library fixed at deployment through `DELEGATECALL`. It uses caller or proxy storage, has no return value or nested calls, and reverts atomically on failure.
- **Manual call-path verification**: The route is nonpayable decoding, `onlyOwner`, internal module dispatch, and then library checkpointing per EID. Owner data cannot select the library. ASI's separate `_validateQuoteSwapPath` failure does not affect this route.

Function: `constructor(address _token, address _lzEndpoint)`

Deployment configures inherited OFT token and endpoint state, decimal conversion, and immutable `nativeToken` and then disables implementation initializers. Its static probes do not authenticate addresses or runtime compatibility.

Inputs

- `_token`
  - **Control**: The deployment mechanism selects `_token` without constructor-level authentication.
  - **Constraints**: The token's `decimals()` function must succeed, decode as `uint8`, satisfy shared-decimal validation, and permit conversion-rate calculation.
  - **Impact**: The address becomes `innerToken`, while its decimals determine conversion behavior without proving TIP-20 compatibility or required roles.
- `_lzEndpoint`
  - **Control**: The deployment mechanism selects `_lzEndpoint`, which is passed unchanged to both inherited bases.
  - **Constraints**: The endpoint's `nativeToken()` function must succeed and decode as an address, but neither endpoint nor returned token is otherwise

validated.

- **Impact**: The address configures inherited messaging, and its reported token becomes immutable `nativeToken` for later fee handling.

Branches and code coverage

Intended branches

- Valid token decimals, conversion, and endpoint native-token data permit deployment; proxy storage initializes while implementation storage remains locked.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Token_returnsExpected`, `FraxOFTMintableAdapterUpgradeableTIP20Test.test_Implementation_CannotBeInitialized`)
- The endpoint's `nativeToken()` function may return zero or another unsuitable address without preventing deployment.
  - ☐ Test coverage
- A token-like contract may pass the `decimals()` probe despite lacking compatible transfer, burn, mint, or role behavior.
  - ☐ Test coverage

Negative behavior

- A creation attempt with nonzero `msg.value` reverts under the nonpayable constructor guard.
  - ☐ Negative test
- A failed, out-of-gas, malformed, invalid-decimal, or overflowing `_token.decimals()` path reverts creation atomically.
  - ☐ Negative test
- A failed, out-of-gas, malformed, or nondecodable `_lzEndpoint.nativeToken()` path reverts creation atomically.
  - ☐ Negative test
- Any constructor failure rolls back prior creation-frame state and logs except consumed gas.
  - ☐ Negative test

Function call analysis

- Constructor trace: The `OFTAdapterUpgradeable(_token, _lzEndpoint)` base constructor statically calls `IERC20Metadata(_token).decimals()`, validates decimals, and configures `innerToken`, conversion, and endpoint state. The

`TempoAltTokenBase(_lzEndpoint)` base constructor then statically calls `IEndpointV2Alt(_lzEndpoint).nativeToken()` and stores the decoded immutable. Finally, `_disableInitializers()` disables implementation initialization. Both external calls send zero value, permit only read-only callee behavior and nested staticcalls, and propagate failures to the creation frame.

- **Manual call-path verification**: The constructor performs no token movement, LayerZero messaging, runtime callback, value-bearing call, or runtime override invocation, and successful construction returns deployed bytecode.

Findings

- The token and endpoint remain deployment trust assumptions because the two ABI probes establish only limited interface liveness, not canonical identity, TIP-20 behavior, roles, messaging compatibility, or fee-token suitability.
- Parser errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern runtime `_quote` expressions and do not justify omitting either resolved constructor-time staticcall.

Function: `initialize(address _delegate)`

The first eligible proxy caller chooses `_delegate`. The initializer sets that address as the endpoint delegate and, after temporarily assigning ownership to the caller, as the final owner. Deployment must perform initialization atomically. The implementation is initializer-locked, and the initializer locally permits the zero address if the endpoint accepts it.

Inputs

- `_delegate`
  - **Control**: The first eligible proxy caller selects `_delegate` without authentication, and it is forwarded unchanged to the fixed endpoint and final ownership.
  - **Constraints**: The `initializer` life cycle requires version 1 to remain available, and the nonpayable check requires `msg.value` to be zero. There is no local nonzero check or `tx.origin` check.
  - **Impact**: On success, the caller becomes the transient owner before `_delegate` becomes the endpoint delegate and final owner. An unauthorized first caller can take control by selecting `_delegate`, while an accepted zero address can disable administration.

Branches and code coverage

Intended branches

- Eligible proxy initialization assigns the supplied delegate as final owner.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Owner_returnsExpected`)
- The endpoint records `_delegate`, and ownership performs both intermediate transitions.
  - ☐ Test coverage
- Endpoint interaction success: the `endpoint.setDelegate(_delegate)` call succeeds before ownership writes, and its return data is neither declared nor decoded.
  - ☐ Test coverage
- Implementation-instance boundary: the `_disableInitializers()` function locks the implementation, so direct initialization reverts.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Implementation_CannotB`
- Zero-value call boundary: a zero-value call can execute, but no native value is forwarded or otherwise used.
  - ☐ Test coverage

Negative behavior

- A consumed proxy initialization state rejects another initialize call.
  - ☑ Negative test (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Proxy_CannotBeReiniti`
- Ordering before endpoint and ownership effects is not independently asserted.
  - ☐ Negative test
- Nonzero native value: any positive `msg.value` reverts at the nonpayable boundary before body execution.
  - ☐ Negative test
- Unauthorized first caller: no authorization protects the first eligible call, so unsafe deployment can let an attacker select `_delegate`.
  - ☐ Negative test
- Endpoint failure: endpoint revert or out-of-gas failure atomically rolls back initialization bookkeeping, endpoint state, ownership, and events.
  - ☐ Negative test
- Zero delegate: endpoint rejection reverts atomically, while endpoint acceptance leaves the final local owner as `address(0)`.
  - ☐ Negative test
- Endpoint callback or other unusual control flow: the trusted endpoint receives control before ownership setup, recursive initialization is rejected, and callback failure rolls back the transaction.

- ☐ Negative test

Function call analysis

- `this.__OFTAdapter_init(_delegate) -> this.__OFTCore_init(_delegate) -> this.__OAppCore_init(_delegate) -> this.__OAppCore_init_unchained(_delegate) -> this.endpoint.setDelegate(_delegate)`
    - Only `endpoint.setDelegate(_delegate)` is external: the fixed endpoint sees the proxy as caller, receives no value, and returns no decoded data, and any failure or unusual control flow reverts atomically.
- **Manual call-path verification**: The generated `this.*` helpers are internal. ASI omitted initializer bookkeeping, `__Ownable_init()`, and `_transferOwnership(_delegate)`. This initializer never calls `__EIP712_init`, so it does not exercise EIP-712 initialization behavior. The ASI errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern separate quote-path expressions.

Function: `recoverERC20(address _tokenAddress, uint256 _tokenAmount)`

Owner-only ERC-20 recovery transfers the requested amount from the adapter/proxy to `owner()` through SafeERC20 and then emits `RecoveredERC20`. Any token, including `innerToken`, may be selected, and the event reports requested values rather than measured balance changes.

Inputs

- `_tokenAddress`
    - **Control**: The authenticated owner selects any ABI-valid token address.
    - **Constraints**: There is no zero-address or `innerToken` restriction, but the target must contain code and return empty data or ABI-encoded `true`.
    - **Impact**: The address determines the external target and emitted token, while malicious or fee-charging tokens may produce different balance changes than reported.
- `_tokenAmount`
    - **Control**: The authenticated owner selects the full-width `uint256` amount.
    - **Constraints**: The value is forwarded unchanged without local bounds or balance checks, and zero is locally permitted.
    - **Impact**: The requested amount is transferred and emitted unchanged, although actual movement remains token-specific.

Branches and code coverage

Intended branches

- The `onlyOwner` modifier succeeds when `msg.sender == owner()`, and `owner()` is also read as the recovery recipient.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_RecoverERC20_onlyOwner`
- A standard TIP-20 transfer returning `true` moves the requested amount to the owner and emits the requested token and amount.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_RecoverERC20_onlyOwner` `FraxOFTMintableAdapterUpgrade-ableTIP20Test.test_RecoverERC20_emitsEvent`)
- SafeERC20 acceptance of empty return data remains untested.
  - ☐ Test coverage
- Zero `_tokenAmount` follows the normal call-and-event path unless rejected by the token.
  - ☐ Test coverage
- A malicious token may report success without moving the requested amount, while the event still records that amount.
  - ☐ Test coverage

Negative behavior

- A caller other than the owner reverts.
  - ☑ Negative test (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_RecoverERC20_onlyOwner`
- Reversion before token interaction and event emission is not independently instrumented.
  - ☐ Negative test
- Nonzero `msg.value` is rejected because the function is nonpayable.
  - ☐ Negative test
- A target without code or a call returning a revert, malformed data, or `false` causes the transaction to revert without a persistent event.
  - ☐ Negative test
- Token-specific rejection, including insufficient balance or a disallowed zero transfer, reverts the entire transaction.
  - ☐ Negative test

Function call analysis

- `SafeERC20.safeTransfer(IERC20(_tokenAddress), this.owner(), _tokenAmount)`
    - SafeERC20 makes one zero-value low-level CALL to `_tokenAddress` for `transfer(owner(), _tokenAmount)`; the token controls execution and return data, may call back or reenter before emission, and all nested effects roll back if the outer call reverts.
- **Manual call-path verification**: The generated `this.owner()` is incorrect because source `owner()` is an inherited internal getter and storage read. The only external edge is the token `CALL`. Builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern other functions.

Function: `send(SendParam _sendParam, MessagingFee _fee, address _refundAddress)`

The permissionless `send` function rate-limits and burns TIP-20 principal, prepays fees, submits a packet, returns receipts, and emits `OFTSent`.

Inputs

- `_sendParam`
    - **Control**: The caller selects destination, recipient, amount, minimum, options, composition, and unused `oftCmd`.
    - **Constraints**: Normalization, minimums, limits, permissions, peers, options, and inspection constrain execution.
    - **Impact**: It determines principal, packet, receipt, and event; zero-low-160-bit recipients pass validation, and `_toSD` may understate packet amount after full local burn and accounting.
- `_fee`
    - **Control**: The caller supplies unbound `nativeFee` and `lzTokenFee` values.
    - **Constraints**: Balances, allowances, routing, liquidity, wrapping, and endpoint acceptance constrain payment.
    - **Impact**: It prepays native and optional LZ-token fees without letting callers bind the fee token or an absolute input cap.
- `_refundAddress`
    - **Control**: The caller selects any refund address, including zero.
    - **Constraints**: The endpoint controls refund acceptance and delivery.
    - **Impact**: This address receives endpoint excess, while swap leftovers return to `msg.sender`.
- Implicit execution inputs

- **Control**: The caller controls funding; configuration fixes tokens, peers, limits, inspector, endpoint, and libraries.
- **Constraints**: The call must carry zero `msg.value`, dependencies must be usable, and balances and allowances must be sufficient.
- **Impact**: The caller (`msg.sender`) funds execution, `block.timestamp` drives decay, and `tx.origin` remains unused.

Branches and code coverage

Intended branches

- Zero `msg.value` passes the entry check.

  ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_SucceedsWhenMsgVa`

- Dust-normalized principal proceeds when satisfying `minAmountLD`.

  ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_SucceedsWhenMsgVa`

- With rate limiting disabled by default, the tested nonzero send skips capacity consumption.

  ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_SucceedsWhenMsgVa`

- A zero amount also skips consumption; otherwise enabled capacity decays and is consumed.

  ☐ Test coverage

- A successful debit increments transfer telemetry and burns the principal.

  ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Debit_BurnsTokensAndTr`

- The ordering of telemetry before transfer and burn is not instrumented.

  ☐ Test coverage

- An empty composition sends successfully without compose data.

  ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_SucceedsWhenMsgVa`

- A nonempty composition encodes `msg.sender` and the payload.

  ☐ Test coverage

- Enforced and caller options combine, followed by optional inspector approval.

  ☐ Test coverage

- Zero `nativeFee` skips fee-token, DEX, approval, and wrapping operations.

  ☐ Test coverage

- Whitelisted fee input uses direct transfer and wrapping.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_LzSend_NoSwap_WhenUser`)
- Tested nonwhitelisted TIP-20 inputs use a viable bounded DEX route, refund any unspent input through the net balance result, wrap pathUSD, and leave no adapter residue or DEX allowance.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_LzSend_SwapsInnerToken` `FraxOFTMintableAdapterUpgrade-ableTIP20Test.test_LzSend_SwapsAnyTip20ToPathUsd`)
- Cheapest-route selection among multiple viable targets is not isolated.
  - ☐ Test coverage
- A zero `lzTokenFee` permits the send without an LZ-token payment.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_SucceedsWhenMsgVa`)
- A positive LZ-token fee transfers the endpoint-selected token.
  - ☐ Test coverage
- A configured peer and accepting endpoint return the asserted messaging and OFT receipts.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_SucceedsWhenMsgVa`)
- Success emits `OFTSent`.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` reverts with `OFTAltCore__msg_value_not_zero`.
  - ☑ Negative test (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_RevertsWhenMsgVal`)
- Precedence before debit, fee interaction, and logs is not independently asserted.
  - ☐ Negative test
- Normalized amounts below `minAmountLD` revert before principal movement.
  - ☐ Negative test
- Insufficient capacity reverts; sends can contend for capacity, and timestamp-based recovery is floored.
  - ☐ Negative test
- Supply tracking permits negative supply but reverts on checked overflow.
  - ☐ Negative test

- Principal transfer or burn failures abort the send.
  - ☐ Negative test
- Invalid options, inspector rejection, or missing peers revert atomically.
  - ☐ Negative test
- Missing routes or failed token, DEX, wrapper, or LZ-token operations revert.
  - ☐ Negative test
- Nonwhitelisted `nativeFee` above `uint128` truncates for swapping, so wrapping the full fee can fail.
  - ☐ Negative test
- Synchronous endpoint failure rolls back; asynchronous failure cannot restore finalized principal.
  - ☐ Negative test
- Recipients with zero low 160 bits pass validation.
  - ☐ Negative test
- The `_toSD` function truncates the packet amount while local burn and accounting remain full.
  - ☐ Negative test
- Zero refund addresses remain accepted, and callers cannot bind the fee token or an absolute input cap.
  - ☐ Negative test

Function call analysis

- `this._debit(...) -> this._consumeOutboundRateLimit(...) -> RateLimiterLib.consumeRateLimit(..., True)`
  - This is a linked-library `DELEGATECALL`. `msg.sender` and `address(this)` are preserved, and storage is the host's RateLimiter slot. It may revert with `RateLimitExceeded`.
- `this._debit(...) -> ITIP20(address(this.innerToken)).transferFrom(msg.sender, address(this), amountSentLD)`
  - This external `CALL` goes to the TIP20 token, which is trusted. It pulls tokens from the sender into the adapter and requires prior allowance.
- `this._debit(...) -> ITIP20(address(this.innerToken)).burn(amountSentLD)`
  - This external `CALL` goes to the TIP20 token. It burns from the adapter's own balance and needs the adapter to hold `ISSUER_ROLE`. It reverts if the role is missing.

- `this._buildMsgAndOptions(...) -> OFTMsgCodec.encode(..., this._toSD(_amountLD), ...)`
    - This is pure encoding. `_toSD` downcasts to `uint64` after dividing by `decimalConversionRate`.
- `this._buildMsgAndOptions(...) -> IOAppMsgInspector($.msgInspector).inspect(message, options)`
    - This is an optional external `STATICCALL`. It only happens when `msgInspector != address(0)`, and the call is `view`. The owner sets the inspector, and it can revert to block the send.
- `this._lzSend(...) -> this._payLzToken(...) -> this.endpoint.lzToken()`
    - This path runs only when `_fee.lzTokenFee > 0`. It is an external `STATICCALL` to the LayerZero endpoint, and the result is deliberately not cached because the endpoint can change its `lzToken`.
- `this._lzSend(...) -> SafeERC20.safeTransferFrom(IERC20(lzToken), msg.sender, address(this.endpoint), ...)`
    - This is an external `CALL` to the endpoint-reported `lzToken`, which is trusted only as far as the endpoint is. It moves the LZ token directly from the user to the endpoint and does not pass through the adapter
- `this._lzSend(...) -> this.endpoint.send{value: messageValue}(..., _refundAddress)`
    - This is the final effectful external `CALL` in the function. `messageValue` is always `0`, because `_payNative` is overridden and returns `0` after paying in ERC20.
- External edges include fixed-library delegatecalls, inspector staticcall, and unguarded token, DEX, wrapper, and endpoint callbacks; synchronous failures revert.
- **Manual call-path verification**: The actual order is `_debit` → build/inspect → `_payNativeAltToken` → optional LZ-token payment → `endpoint.send`; trace labels misclassify internals and omit `_payNative`, fee routing uses `userTokens(sender)` / `PATH_USD`, and sibling `_quote` parsing fails at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143`.

Function: `setAllowNegativeSupply(uint32 _eid, bool _allow)`

This owner-only setter writes `allowNegativeSupply[_eid]` and emits `SetAllowNegativeSupply`, even when unchanged. It neither inspects nor repairs counters; the flag controls later transfer-from enforcement.

Inputs

- `_eid`

    - **Control**: The current owner supplies `_eid` through calldata.
    - **Constraints**: ABI decoding enforces `uint32`, but no EID, peer, initialization, or ledger-state validation occurs.
    - **Impact**: It selects the mapping entry to update and emit, without affecting other EIDs.
- `_allow`
    - **Control**: The current owner supplies `_allow` through calldata.
    - **Constraints**: ABI decoding enforces a canonical boolean, with no equality or ledger-state check.
    - **Impact**: Setting `false` enables later nonnegative-supply enforcement, while setting `true` bypasses only that comparison without changing counters or checked arithmetic.

Branches and code coverage

Intended branches

- An authorized call with `_allow == true` writes `true` and emits the event, while later counter arithmetic remains checked.
    - ☐ Test coverage
- An authorized call with `_allow == false` on a nonnegative ledger writes `false` and enables the later transfer-from inequality check.
    - ☐ Test coverage
- An authorized call with `_allow == false` on a negative ledger succeeds; the component-level 1/1 probe confirms continued `totalSupply` reversion and later transfer-from blocking, but is not target-specific.
    - ☐ Test coverage
- Supplying the already stored value still performs the assignment and emits the event because there is no equality guard.
    - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts in `onlyOwner` before any write or event.
    - ☐ Negative test
- Noncanonical or out-of-range ABI calldata is rejected before the function body executes.
    - ☐ Negative test
- Any nonzero native value is rejected by the nonpayable dispatcher before execution.
    - ☐ Negative test

- A negative ledger does not make this setter revert, though later `totalSupply` or transfer-from enforcement may revert.
  - ☐ Negative test
- Exceptional failure, including insufficient gas or static execution, rolls back both the assignment and event.
  - ☐ Negative test

Function call analysis

- `this._setAllowNegativeSupply(_eid, _allow) -> this._getSupplyTrackingStorage()`
  - Both calls are internal, and the Yul helper only assigns the fixed storage slot before the mapping write and event.
- **Manual call-path verification**: The generated `this.*` trace is incorrect and omits `SetAllowNegativeSupply`. There are no external calls, callbacks, approvals, token or native movements, or message encoding. The parser failure at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` concerns unrelated `_payNativeAltToken(_nativeFee, address(endpoint))` code.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This owner-only entry point replaces the default rate-limit configuration through a fixed-library delegatecall in caller storage. It emits `DefaultRateLimitConfigSet` without checkpointing EID state.

Inputs

- `_defaultConfig`
  - **Control**: The owner controls all seven struct members, while authorization, the linked library address, and the storage namespace are not caller-selectable.
  - **Constraints**: Enabled directions require nonzero limits and windows, while disabled fields and `overrideDefaultConfig` are unrestricted within their ABI types.
  - **Impact**: Success replaces the complete default configuration, affecting non-overridden EIDs without checkpointing their existing states and moving no tokens or native value.

Branches and code coverage

Intended branches

- The current owner passes `onlyOwner` and forwards `_defaultConfig` to the internal module helper.
  - ☐ Test coverage
- Enabled outbound limiting requires nonzero `outboundLimit` and `outboundWindow`, while disabled outbound limiting imposes no such validation.
  - ☐ Test coverage
- Enabled inbound limiting requires nonzero `inboundLimit` and `inboundWindow`, while disabled inbound limiting imposes no such validation.
  - ☐ Test coverage
- Successful validation replaces `defaultConfig` and emits flags, limits, and windows but not `overrideDefaultConfig`.
  - ☐ Test coverage
- EIDs without per-EID overrides use the replacement default, while overridden EIDs retain their stored configurations.
  - ☐ Test coverage
- Later decay uses replacement parameters from the prior `lastUpdated`, with the 1/1 reproduction showing `1,000` uncheckpointed versus `910` checkpointed units.
  - ☐ Test coverage
- The path has no loop, virtual dispatch, token or native-value movement, LayerZero message, callback, or return-data branch.
  - ☐ Test coverage

Negative behavior

- A non-owner call reverts before invoking the helper or changing configuration.
  - ☐ Negative test
- Malformed calldata or noncanonical ABI values revert during decoding before the function body.
  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable dispatcher without retaining or forwarding value.
  - ☐ Negative test
- Enabled outbound limiting with a zero limit or window reverts with `InvalidRateLimitConfig` before writing or emitting.
  - ☐ Negative test

- Enabled inbound limiting with a zero limit or window reverts with `InvalidRateLimitConfig` before writing or emitting.
  - ☐ Negative test
- Any linked-library delegatecall failure propagates and atomically rolls back storage changes and logs.
  - ☐ Negative test

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`
  - ASI adds `this.` to the direct internal helper call. The fixed-target library delegatecall preserves caller context, validates the owner-selected struct, writes caller storage, returns no value, and exposes no further external call or reentrancy point.
- **Manual call-path verification**: The verified path is `FraxOFTMintableAdapterUpgradeableTIP20.setDefaultRateLimitConfig` → internal `RateLimiterModule._setDefaultRateLimitConfig` → delegatecall to `RateLimiterLib.setDefaultRateLimitConfig`, which writes slot `0xf02f50e1c246a23e457c592e128a403f9c1cc60baa3a1ad2dc69bb4a5a51be00`; parser failures at `_payNativeAltToken(...)` and `FraxOFTUpgradeableTempo._validateQuoteSwapPath(...)` are off path.

Function: `setFeeSwapSlippageBps(uint16 _bps)`

This owner-only, nonpayable setter forwards `_bps` unchanged, rejects values above 200, stores the raw `uint16` in fixed namespaced storage, and then emits `FeeSwapSlippageBpsSet`. A stored zero is exposed as the 50 bps default and is consumed only by later nonwhitelisted fee-token paths.

Inputs

- `_bps`
  - **Control**: The caller selects `_bps`, but only the current owner may apply it through authorization based on `msg.sender`.
  - **Constraints**: Valid `uint16` values from 0 through 200 are accepted, with zero selecting the 50 bps default, while nonzero `msg.value`, unauthorized callers, and values above 200 revert.
  - **Impact**: On success, the function stores and emits the raw value. Later nonwhitelisted fee-token paths use its effective value for slippage padding, while any revert preserves the prior configuration.

Branches and code coverage

Intended branches

- An authorized owner with `_bps == 0`: stores and emits zero, while subsequent reads return the 50 bps default.
  - ☐ Test coverage
- An authorized owner with `0 < _bps < MAX_FEE_SWAP_SLIPPAGE_BPS`: stores and emits the requested value from 1 through 199.
  - ☐ Test coverage
- An authorized owner with `_bps == MAX_FEE_SWAP_SLIPPAGE_BPS`: accepts, stores, and emits the 200 bps boundary.
  - ☐ Test coverage

Negative behavior

- The caller is not the owner (`msg.sender != owner()`): the `onlyOwner` modifier reverts before any storage mutation or event.
  - ☐ Negative test
- Value above the maximum (`_bps > MAX_FEE_SWAP_SLIPPAGE_BPS`): the setter reverts with `FeeSwapSlippageTooHigh(_bps)` before writing or emitting.
  - ☐ Negative test
- For nonzero `msg.value`, Solidity rejects the nonpayable call before executing the modifier or body.
  - ☐ Negative test

Function call analysis

- `this._setFeeSwapSlippageBps(_bps) -> this._getAltTokenStorage()`
  - The generated `this.*` notation is inaccurate because both calls resolve internally, and the assembly helper only selects the fixed `AltTokenStorageLocation`.
- **Manual call-path verification**: This setter has no external call, transfer, approval, callback, return data, or reentrancy point. Downstream nonwhitelisted paths apply the effective bps in `uint256`, require one extra input unit, and saturate at the `uint128` maximum. Whitelisted 1:1 wraps bypass it. The bounded per-instance tolerance limits divergence, but fixed OFT sends cannot bind execution to a sender-selected maximum. The ASI builder failure at `contracts/FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` concerns the separate `_payNative` path.

Function: `setInitialTotalSupply(uint32 _eid, uint256 _amount)`

This owner-only setter forwards `_eid` and `_amount` unchanged to the inherited internal helper, overwriting the namespaced per-EID baseline and emitting `SetInitialTotalSupply`. It moves no tokens but can create accounting states that later underflow or violate transfer bounds.

Inputs

- `_eid`
  - **Control**: The caller selects `_eid`, but only `msg.sender == owner()` can reach the state change.
  - **Constraints**: ABI decoding requires a canonical `uint32`, with no configured-EID or nonzero check.
  - **Impact**: The value keys `initialTotalSupply` and is emitted unchanged as the indexed event EID.
- `_amount`
  - **Control**: The caller selects `_amount`, but only the owner can store and emit it.
  - **Constraints**: Any `uint256` is accepted without ledger or token-supply validation, while successful execution requires `msg.value == 0`.
  - **Impact**: The value overwrites the baseline and can make derived supply underflow or later negative-supply bounds revert.

Branches and code coverage

Intended branches

- An authorized caller writes the selected baseline, which is returned by `initialTotalSupply(_eid)`.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_SetInitialTotalSupply_`
- Success emits `SetInitialTotalSupply` and returns no ABI data.
  - ☐ Test coverage
- All ABI-valid `_eid` and `_amount` values, including zero and existing entries, follow the same write-and-emit path.
  - ☐ Test coverage

Negative behavior

- A non-owner caller reverts.
  - ☑ Negative test (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_SetInitialTotalSupply_`

- Ordering before writes and events is not independently asserted.
  - ☐ Negative test
- Nonzero value or malformed calldata reverts before effects.
  - ☐ Negative test
- Unknown EIDs and ledger-inconsistent or zero amounts are accepted. Failures appear only in later accounting operations.
  - ☐ Negative test
- Any authorization or subsequent revert atomically removes the baseline update and event.
  - ☐ Negative test

Function call analysis

- `this._setInitialTotalSupply(_eid, _amount) -> this._getSupplyTrackingStorage()`
  - The generated `this.*` notation represents internal calls; the fixed-slot assembly only returns namespaced storage before the helper writes the baseline and emits the event.
- **Manual call-path verification**: The actual path is `onlyOwner` → internal `_setInitialTotalSupply` → private `_getSupplyTrackingStorage` → write → emit, with no external call, callback, reentrancy point, or value/token movement; the parser failure at line 174 concerns the separate nested `_quote` expression and does not affect this setter.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This owner-only setter checkpoints each EID under its previously effective configuration, stores replacements, and emits checkpoint/configuration events. Repeated EIDs are order sensitive, while empty arrays succeed without effects.

Inputs

- `_params`
  - **Control**: The owner controls the ordered array of EIDs and configurations.
  - **Constraints**: Enabled directions require nonzero limits and windows, while disabled fields may be zero.
  - **Impact**: Each element checkpoints old-effective usage before replacement, and the last repeated EID determines final storage.
- Implicit state and environmental inputs

- **Control**: Execution depends on the owner, existing/default rate-limit state, gas, timestamp, and linked library address.
- **Constraints**: The `block.timestamp` value is narrowed to `uint32` under the pre-2106 operating assumption.
- **Impact**: The old effective configuration determines decay, while delegatecall updates the OFT or proxy's namespaced storage.

Branches and code coverage

Intended branches

- Authorized entry: the current owner passes `onlyOwner` and invokes the internal helper once.
  - ☐ Test coverage
- Empty array: an empty array succeeds without writes, events, or return data.
  - ☐ Test coverage
- Valid outbound configuration: outbound validation passes when disabled or when both limit and window are nonzero.
  - ☐ Test coverage
- Valid inbound configuration: inbound validation passes when disabled or when both limit and window are nonzero.
  - ☐ Test coverage
- Initial or nonadvancing checkpoint: zero or nonadvancing timestamps preserve usage and update `lastUpdated`.
  - ☐ Test coverage
- Elapsed checkpoint with inactive zero fields: zero old usage, limit, or window preserves that direction's usage.
  - ☐ Test coverage
- Full-window decay: elapsed time at least equal to the old window clears that direction's usage.
  - ☐ Test coverage
- Partial-window decay: usage decreases by `oldLimit * elapsed / oldWindow`, floored at zero.
  - ☐ Test coverage
- Configuration replacement and events: each valid element emits `RateLimitCheckpointed` before storing and emitting `RateLimitConfigSet`.
  - ☐ Test coverage
- Default-versus-override selection: checkpointing uses the stored override when enabled and otherwise uses the current default.

☐ Test coverage

- Repeated EID with override enabled: a later duplicate checkpoints against the preceding stored override.

☐ Test coverage

- Repeated EID with override disabled: a later duplicate checkpoints against the current default, and the final occurrence controls storage.

☐ Test coverage

- Disable/reenable transition: zero inactive limit or window preserves usage, while retained nonzero values permit checkpoint decay; matched tests pass 7/7.

☐ Test coverage

- Successful completion: after all elements are processed, the library and wrapper return no data.

☐ Test coverage

- Coverage status: source review found 6/6 module tests and 7/7 matched transition tests but no exact branch mapping.

☐ Test coverage

Negative behavior

- Unauthorized caller: a non-owner reverts before the helper call, state changes, or events.

☐ Negative test

- Native-value submission: nonzero `msg.value` is rejected by the nonpayable dispatcher.

☐ Negative test

- ABI decoding failure: malformed dynamic-array calldata reverts before state mutation.

☐ Negative test

- Invalid outbound configuration: an enabled outbound direction with a zero limit or window reverts before checkpointing.

☐ Negative test

- Invalid inbound configuration: an enabled inbound direction with a zero limit or window reverts before checkpointing.

☐ Negative test

- Later-element failure: any later validation, delegatecall, or gas failure atomically reverts prior iterations and logs.

☐ Negative test

- Unbounded batch exhaustion: a batch exceeding available gas reverts without partially applying a prefix.

☐ Negative test

- No implicit usage reset: disabling a direction does not itself clear accrued usage.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
  - The real external edge is a linked-library `DELEGATECALL` that preserves caller, value, and contract context; mutates namespaced storage; emits logs; returns no data; and propagates reverts atomically without callbacks.
- **Manual call-path verification**: The concrete path is `setRateLimitConfigs` → internal `_setRateLimitConfigs` → linked `RateLimiterLib.setRateLimitConfigs`; generated `this._setRateLimitConfigs` is incorrect, while the unrelated nested `_quote` ASI parsing failure does not affect this path.

Function: `setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)`

This owner-only setter delegates to the linked library, which replaces namespaced `globalConfig` and emits `RateLimitGlobalConfigSet`. It preserves configurations, usage, and timestamps. Checkpoints remain stateful while disabled, and setting a zero limit or window later preserves usage.

Inputs

- `_globalConfig`
  - **Control**: The owner supplies `_globalConfig.isGloballyDisabled`, which is stored and emitted unchanged.
  - **Constraints**: ABI decoding, canonical boolean encoding, zero `msg.value`, inherited `onlyOwner`, and correct linkage to the fixed RateLimiterLib bytecode are required.
  - **Impact**: Setting `true` bypasses consumption globally, while setting `false` resumes retained enforcement; any failure rolls back the write and event.

Branches and code coverage

Intended branches

- Disable globally: an authorized call with `true` stores and emits `true`. It bypasses consumption and `RateLimitConsumed`, reports unlimited inbound and outbound availability, retains the `quoteOFT` cap, and leaves checkpoint administration active.

- ☐ Test coverage

- Enable globally: an authorized call with `false` stores and emits `false`, resuming enforcement against retained configurations, usage, and timestamps. Later queries decay only in memory, while successful consumption persists refreshed state.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: inherited `onlyOwner` reverts before the helper or library executes, preserving the prior flag and logs.
  - ☐ Negative test
- Malformed struct encoding: invalid calldata or a noncanonical boolean reverts during ABI decoding before any state change.
  - ☐ Negative test
- Native value supplied: the nonpayable entry point rejects nonzero `msg.value` before the function body executes.
  - ☐ Negative test
- Linked-library execution failure: out-of-gas or another delegatecall failure bubbles up and atomically rolls back the storage write and event.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The generated `this.` receiver is incorrect: the source first uses an internal helper, followed by a single fixed-address library `DELEGATECALL` in the caller's storage context. The call has no application return value, callback, token or native movement, or messaging. Malicious or mislinked bytecode could mutate arbitrary caller storage.
- **Manual call-path verification**: The resolved path is ABI decoding and nonpayable dispatch, inherited `onlyOwner`, internal `RateLimiterModule._setRateLimitGlobalConfig`, and then linked `RateLimiterLib.setRateLimitGlobalConfig`. Its failure bubbles up, and its event is emitted from the caller. The unrelated builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` do not alter this path.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

This owner-only setter forwards the array through the inherited module to the linked library. Accepted states reset or seed usage, affecting later decayed capacity under the effective configuration; this privileged effect is not an authorization bypass.

Inputs

- `_params`
  - **Control**: The owner controls array order, repeated EIDs, usage values, and timestamps, with final repeated entries prevailing.
  - **Constraints**: The call must be nonpayable and owner-authorized, while each state requires a nonfuture timestamp and zero usage when its timestamp is zero.
  - **Impact**: Accepted entries overwrite `states[eid]` unchanged and emit `RateLimitStateSet`, while any failure atomically reverts the batch.

Branches and code coverage

Intended branches

- An owner call forwards the complete array unchanged through the inherited module to the linked library.
  - ☐ Test coverage
- An empty array succeeds without writes or `RateLimitStateSet` events.
  - ☐ Test coverage
- Zero timestamp with both usage fields zero is accepted, stored, and emitted.
  - ☐ Test coverage
- A nonzero nonfuture timestamp accepts any `uint112` usage values, including values at or above effective limits.
  - ☐ Test coverage
- Valid entries execute sequentially, and repeated EIDs emit each time while the final entry remains stored.
  - ☐ Test coverage

Negative behavior

- A non-owner call fails `onlyOwner` before library execution, writes, or events.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable dispatcher before the function body executes.

- ☐ Negative test
- Malformed calldata reverts during ABI decoding before authorization or mutation.
  - ☐ Negative test
- A future `lastUpdated` reverts with `InvalidRateLimitState()` before that entry is written.
  - ☐ Negative test
- A zero `lastUpdated` with either usage field nonzero reverts with `InvalidRateLimitState()`.
  - ☐ Negative test
- Any later validation failure bubbles up and rolls back all preceding writes and logs.
  - ☐ Negative test
- Gas exhaustion during a large batch reverts all writes and logs rather than leaving partial state.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
  - The generated `this.` prefix is incorrect: the actual path is internal nonvirtual `RateLimiterModule._setRateLimitStates` followed by linked-library `DELEGATECALL`, which preserves caller context, validates and writes fixed namespaced storage, emits events, returns no data, and makes no further external calls.
- **Manual call-path verification**: The exact path is `setRateLimitStates(_params)` → `RateLimiterModule._setRateLimitStates(_params)` → `RateLimiterLib.setRateLimitStates(_params)` via `DELEGATECALL`; ASI omitted the internal hop, while reported builder failures concern unrelated code.

5.8. Module: FraxOFTUpgradeable.sol

Function: `checkpointRateLimits(uint32[] _eids)`

This owner-only, nonpayable function passes `_eids` unchanged through the inherited helper to a linked-library delegatecall in the OFT or proxy storage context. It decays usage, updates shared `lastUpdated`, and emits `RateLimitCheckpointed` for each entry.

Inputs

- `_eids`
  - **Control**: The owner chooses the array length, order, duplicates, and each `uint32` EID.
  - **Constraints**: Empty arrays and duplicates are allowed, while calldata size and gas bound execution.
  - **Impact**: Each EID keys configuration and state, with duplicates causing repeated writes and events in order.
- `msg.sender`
  - **Control**: The immediate caller controls `msg.sender`, which is preserved through delegatecall.
  - **Constraints**: It must satisfy `onlyOwner`.
  - **Impact**: It affects authorization only and is not read by the library.
- Current owner state
  - **Control**: Ownership state is managed through the inherited ownership life cycle.
  - **Constraints**: The recognized owner must equal `msg.sender` at call time.
  - **Impact**: It gates checkpointing but is not modified.
- `msg.value`
  - **Control**: The caller chooses the attached native value.
  - **Constraints**: The nonpayable entry point requires zero value.
  - **Impact**: No native value reaches the helper or library.
- `tx.origin`
  - **Control**: The originating transaction determines `tx.origin`.
  - **Constraints**: No constraint applies because the call path never reads it.
  - **Impact**: It has no authorization, storage, arithmetic, or interaction effect.
- `block.timestamp`
  - **Control**: The block producer influences the consensus timestamp within protocol bounds.
  - **Constraints**: Conversion to `uint32` relies on migration before the year-2106 boundary.
  - **Impact**: It controls elapsed-time decay and the stored and emitted shared timestamp.
- Rate-limit configuration and state
  - **Control**: Administrative and prior accounting operations determine stored defaults, overrides, and states.
  - **Constraints**: Checkpointing performs no new validation and selects configuration solely by `overrideDefaultConfig`.

    - **Impact**: Numeric limits, windows, usages, and timestamps determine decay, while global and directional flags are ignored.
- Deployment-linked RateLimiterLib address and code
    - **Control**: Deployment-time linking fixes the library target outside caller control.
    - **Constraints**: Correct linking and deployed bytecode are residual deployment assumptions.
    - **Impact**: Delegatecall mutates the calling OFT or proxy's storage, and failure reverts the entire transaction.

Branches and code coverage

Intended branches

- Owner and empty array: an authorized empty array performs no iterations, writes, or events.
    - ☐ Test coverage
- Default-config selection: a non-overridden EID uses `defaultConfig`; changing the default parameters materializes decay under the old parameters, as supported by the 1/1 transition result of `910` rather than `1,000`.
    - ☐ Test coverage
- EID-specific–config selection: an EID with `overrideDefaultConfig == true` uses its stored configuration.
    - ☐ Test coverage
- Uninitialized or nonadvancing timestamp state: a zero or non-earlier `lastUpdated` preserves usage and stores the current `uint32` timestamp.
    - ☐ Test coverage
- Zero-factor decay path: zero usage, limit, or window preserves that direction's usage, consistent with the matched 7/7 zero-value suite.
    - ☐ Test coverage
- Full-window decay path: when elapsed time reaches the selected window, the corresponding nonzero usage becomes zero.
    - ☐ Test coverage
- Partial-window decay path: partial decay subtracts `floor(limit * elapsed / window)` without underflow; shared timestamps and flooring can discard subunit recovery.
    - ☐ Test coverage
- Duplicate EID path: every duplicate rewrites and emits, while later same-transaction occurrences preserve the just-checkpointed usage.

- ☐ Test coverage

- Globally or directionally disabled path: checkpointing ignores global and directional enable flags and still applies numeric limit and window fields.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller failing `onlyOwner` reverts before delegatecall, writes, or events.
  - ☐ Negative test
- Native-value submission: any nonzero `msg.value` is rejected before body execution.
  - ☐ Negative test
- Malformed calldata: invalid dynamic-array encoding cannot reach the function body.
  - ☐ Negative test
- Resource exhaustion or linked-library revert: gas exhaustion or library failure reverts atomically, rolling back all prior per-EID writes and events.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The source calls the inherited helper internally then makes a fixed-target delegatecall. The library returns no value, accesses namespaced storage, emits logs, and makes no callback.
- **Manual call-path verification**: The verified path is `FraxOFTUpgradeable.checkpointRateLimits` → `RateLimiterModule._checkpointRateLimits` → `RateLimiterLib.checkpointRateLimits`; generated `this._checkpointRateLimits` is inaccurate, and ASI failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` are outside this target.

Function: `constructor(address _lzEndpoint)`

The constructor deploys the implementation by forwarding `_lzEndpoint` unchanged to OFTUpgradeable and then disables initializers. This locks the implementation's storage, not the proxy's; the endpoint and imported behavior remain deployment assumptions.

Inputs

- `_lzEndpoint`
  - **Control**: The immediate creator selects `_lzEndpoint`, subject to any external factory policy.
  - **Constraints**: ABI decoding applies, but the target performs no local address, code, interface, allowlist, or chain validation.
  - **Impact**: The value is forwarded unchanged to inherited construction, whose rejection reverts deployment and whose acceptance may configure an incorrect endpoint.

Branches and code coverage

Intended branches

- Base construction completes and installs runtime code used through the initialized proxy.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`, `RateLimiterModuleTest.test_QuoteOFTMax_FollowsDecimalConversionRate`)
- The implementation instance is locked against guarded initialization.
  - ☐ Test coverage
- Endpoint acceptance boundary: the target locally accepts zero, codeless, or incompatible endpoint addresses unless inherited construction rejects them.
  - ☐ Test coverage
- Creation-caller boundary: the constructor performs no caller authorization; any deployment restriction belongs to the creating factory or process.
  - ☐ Test coverage

Negative behavior

- Malformed or noncanonical constructor data fails ABI decoding before deployment.
  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable boundary.
  - ☐ Negative test
- A revert or out-of-gas failure in `OFTUpgradeable(_lzEndpoint)` aborts creation before the constructor body completes.
  - ☐ Negative test
- A revert or out-of-gas failure in `_disableInitializers()` atomically rolls back construction.
  - ☐ Negative test

- Successful deployment locks guarded initialization only for the implementation instance, not a separate proxy.
  - ☐ Negative test
- No branch-level constructor test evidence is identified, so coverage remains unmarked.
  - ☐ Negative test

Function call analysis

- **`OFTUpgradeable(_lzEndpoint)` — inherited base-constructor invocation**: Solidity statically invokes the imported base constructor in the same creation context with `_lzEndpoint` unchanged; no return value is consumed, and any failure atomically aborts deployment.
- **`_disableInitializers()` — inherited internal helper**: This no-argument internal call runs after base construction and locks the implementation instance against guarded initialization. Any failure propagates, and the call moves no value or tokens.
- **Enclosing EVM contract creation**: The creator supplies init code, gas, and zero value; malformed input, dependency failure, or insufficient gas leaves no runtime code or persistent constructor state.
- **Manual call-path verification**: ASI emitted no generated FCA sink header. Manual review confirms ABI/nonpayable entry, static base construction, `_disableInitializers()`, and creation completion. Unrelated builder errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` do not alter this path.

Findings

- No constructor-specific vulnerability or identity collision is established; the principal assumptions are correct endpoint selection, inherited dependency behavior, and separate timely proxy initialization.
- Runtime freezing, fee selection and refunds, rate-limit configuration and recovery, helper initialization, and amount conversion introduce no additional constructor edge cases or vulnerabilities.

Function: `initialize(string _name, string _symbol, address _delegate)`

Any caller can use `reinitializer(3)` while the proxy version is below 3 and initialization is inactive, choosing metadata, endpoint delegate, and final ownership. Deployments must initialize atomically; implementation constructors disable only their own initializers.

Inputs

- `_name`
  - **Control**: The successful initializer caller supplies the EIP-712 and ERC-20 name unchanged.
  - **Constraints**: Any ABI-valid string is accepted, while `version()` is fixed at `"1.2.0"`.
  - **Impact**: It sets token metadata and the EIP-712 domain name used by later signature validation.
- `_symbol`
  - **Control**: The successful initializer caller supplies the ERC-20 symbol unchanged.
  - **Constraints**: Any ABI-valid string is accepted without content or length checks beyond gas limits.
  - **Impact**: It sets ERC-20 symbol metadata but does not affect the EIP-712 domain or endpoint call.
- `_delegate`
  - **Control**: The successful initializer caller supplies the endpoint delegate and final owner.
  - **Constraints**: The wrapper accepts any address, including zero if the configured endpoint permits it.
  - **Impact**: It is passed to `endpoint.setDelegate` and then stored as owner. Before that transfer, `msg.sender` becomes owner.

Branches and code coverage

Intended branches

- With version below 3 and no active initialization, the modifier executes the body and emits `Initialized(3)` on success.
  - ☐ Test coverage
- The initialized EIP-712 name and version produce a domain under which the signed permit succeeds.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- Both `_name` and `_symbol` initialize the exact ERC-20 metadata values.
  - ☐ Test coverage
- The `__OFT_init` function stores metadata and reaches `endpoint.setDelegate(_delegate)` through the OFT/OApp initializer chain.
  - ☐ Test coverage
- Proxy initialization assigns the supplied delegate as final owner, allowing its

owner-gated configuration calls.

- ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `FraxOFTUpgradeableTest.test_Permit_succeeds`)

- Ownership first passes through the initializing caller after endpoint delegate setup.
  - ☐ Test coverage
- Empty or unusual ABI-valid `_name` and `_symbol` values are accepted subject to gas and downstream success.
  - ☐ Test coverage
- The FraxOFTUpgradeableTempo contract inherits this initializer unchanged, and its send/payment overrides are not dispatched here.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` is rejected by the nonpayable ABI entry point before initialization begins.
  - ☐ Negative test
- Version 3 or greater, or an active initialization context, causes `reinitializer(3)` to revert before the body.
  - ☐ Negative test
- An endpoint revert aborts before ownership initialization and rolls back all local and endpoint effects.
  - ☐ Negative test
- Any later revert also rolls back the successful endpoint update, metadata, ownership logs, and initializer version.
  - ☐ Negative test
- Because no caller-role check exists, an uninitialized proxy can be captured by an untrusted caller.
  - ☐ Negative test
- A zero delegate reverts only if the endpoint or another operation rejects it; otherwise, initialization may finish ownerless.
  - ☐ Negative test

Function call analysis

- `this.__EIP712_init(_name, this.version()) -> this.__EIP712_init_unchained(name, version) -> this._getEIP712Storage()`

    - This chain is internal despite FCA's `this.` notation; `__EIP712_init` lacks `onlyInitializing`, but this call executes within the `reinitializer(3)` context and the helper is absent from the external ABI.
- `this.__OFT_init(_name, _symbol, _delegate) -> this.__OFTCore_init(_delegate) -> this.__OAppCore_init(_delegate) -> this.__OAppCore_init_unchained(_delegate) -> this.endpoint.setDelegate(_delegate)`
    - The `endpoint.setDelegate(_delegate)` call is the sole runtime external call, carries no value, consumes no return value, and may call back while metadata is initialized but ownership is not.
- **Manual call-path verification**: FCA omits metadata initialization, both ownership writes, and `Initialized(3)`; parser failures at `FraxOFTUpgradeableTempo.sol@48` and `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` concern unreachable payment paths.

Function: `initializeV120()`

This function reinitializes the proxy's EIP-712 domain from its stored ERC-20 name and fixed version `"1.2.0"`, clearing cached hashes while leaving other token state unchanged. Any caller may consume an eligible `reinitializer(4)` transition.

Branches and code coverage

Intended branches

- Eligible proxy success path: with an initialized version below 4 and no active initialization, the function writes the domain, exits initialization, and emits `Initialized(4)`.
    - ☐ Test coverage
- Storage-backed name resolution: the `name()` function reads the proxy's stored ERC-20 name, while `version()` returns fixed `"1.2.0"` and neither value is caller-selected.
    - ☐ Test coverage
- EIP-712 storage propagation: the helper writes `_name` and `_version` and zeros `_hashedName` and `_hashedVersion`, affecting later permit and EIP-3009 domain validation.
    - ☐ Test coverage
- Atomic completion: initializer state, EIP-712 writes, and `Initialized(4)` complete together or revert all changes without external interaction.
    - ☐ Test coverage

Negative behavior

- If the initialized version is already 4 or greater, `reinitializer(4)` reverts before any state change persists.
  - ☐ Negative test
- If initialization is already active, the modifier rejects nested execution without persisted changes.
  - ☐ Negative test
- Direct implementation calls fail because its constructor disables initializers, leaving intended execution to eligible proxy storage.
  - ☐ Negative test
- Calling this on a proxy below version 3 can advance it directly to 4 and permanently block the later `reinitializer(3)` step.
  - ☐ Negative test
- On a version-3 proxy, any caller can choose when to consume version 4 and, if the written name or version differs, invalidate signatures for the prior domain; the caller cannot select either string or gain token authority.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by nonpayable dispatch, with no ETH retained or state changes persisted.
  - ☐ Negative test
- Caller identity alone never causes rejection because the function has no owner or service-admin authorization check.
  - ☐ Negative test
- No supplied evidence executes a specific branch, so all coverage markers remain unchecked.
  - ☐ Negative test

Function call analysis

- `this.__EIP712_init(this.name(), this.version()) -> this.__EIP712_init_unchained(name, version) -> this._getEIP712Storage()`
  - These are internal getter/helper calls ending in a fixed-slot Yul storage-pointer assignment, with no external edge, callback, reentrancy point, or return value.
- **Manual call-path verification**: The generated `this.` notation depicts external dispatch and omits `reinitializer(4)`. The helpers lack `onlyInitializing` but are internal and reached here through the guarded wrapper. Unrelated Tempo parser failures do not affect this path.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This owner-only setter passes the config unchanged to a fixed linked library, which validates it, writes it to the current contract's storage, and emits `DefaultRateLimitConfigSet`. It does not checkpoint EIDs, so replacing the defaults redecays existing usage under the new slope.

Inputs

- `_defaultConfig`
  - **Control**: The owner supplies all seven typed fields, while the linked-library target is fixed and `msg.value` must be zero.
  - **Constraints**: Enabled directions require nonzero limits and windows, disabled directions accept any typed values, and no transition or checkpoint guard exists.
  - **Impact**: Validation stores all seven fields, while the event omits `overrideDefaultConfig` and per-EID default selection depends on each EID's override flag.

Branches and code coverage

Intended branches

- Valid owner input stores an effective default used by later outbound and inbound accounting.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- Success stores all seven fields and emits the six-field `DefaultRateLimitConfigSet` event.
  - ☐ Test coverage
- Disabled tested directions accept zero limits and windows while the enabled opposite direction remains enforceable.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- Disabled directions also accept nonzero inactive limits and windows, which remain stored.
  - ☐ Test coverage
- Default replacement without checkpointing: replacement defaults redecay uncheckpointed usage under the new slope, with the 1/1 reproduction measuring `1,000`

available versus `910` after a pre-change checkpoint.

- ☐ Test coverage — the supplied 1/1 reproduction establishes the component behavior (`1,000` available without the pre-change checkpoint versus `910` with it), but the evidence does not identify an invocation of this exact external declaration in FraxOFTUpgradeable.sol.

- Event projection: successful storage includes all seven fields, but the event omits `overrideDefaultConfig`.
  - ☐ Test coverage — no exact event assertion for this target declaration was supplied.

Negative behavior

- Unauthorized caller: a failed `onlyOwner` check reverts before the helper, storage write, or event.
  - ☐ Negative test — no exact unauthorized-call test for this target declaration was identified in the supplied evidence.
- Invalid enabled outbound configuration: an enabled outbound direction with a zero limit or window reverts with `InvalidRateLimitConfig()`.
  - ☐ Negative test — no named test was mapped to this exact branch through the target wrapper.
- Invalid enabled inbound configuration: after outbound validation, an enabled inbound direction with a zero limit or window reverts with `InvalidRateLimitConfig()`.
  - ☐ Negative test — no named test was mapped to this exact branch through the target wrapper.
- Native-value rejection: nonzero `msg.value` reverts because the declaration is nonpayable.
  - ☐ Negative test — no exact nonzero-value test for this declaration was supplied.
- Malformed calldata: calldata that cannot decode as `RateLimitConfig` reverts during external dispatch.
  - ☐ Negative test — no exact malformed-calldata test was supplied.
- Linked-library failure: a failed or reverting library `DELEGATECALL` bubbles and atomically removes writes and logs.
  - ☐ Negative test — no exact linked-library-failure case was supplied.
- No transition guard: the call replaces the default without checkpointing affected EIDs under the former slope.
  - ☐ Negative test — the verified component-level reproduction demonstrates that replacing the default without checkpointing affected EIDs redecays uncheckpointed usage under the new slope, but no exact negative-test mapping to this target wrapper was supplied.

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`
    - The trace misrenders the direct nonvirtual internal helper as `this.`. The fixed linked library then executes by `DELEGATECALL`, preserving the sender and using current-context storage. It returns no value, makes no further external call, and propagates failures atomically.
- **Manual call-path verification**: Source confirms direct internal dispatch to the inherited helper and fixed library endpoint. The library validates, stores, and emits without token movement, native transfer, callbacks, or reentrancy.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This owner-only setter forwards `_params` unchanged through the inherited helper to the linked library by `DELEGATECALL`. Each element validates, checkpoints both directions under the old effective config, stores the replacement, and emits `RateLimitCheckpointed` and then `RateLimitConfigSet`.

Inputs

- `_params`
    - **Control**: The owner controls array length, order, duplicates, EIDs, and all configuration fields, while the linked target and selector are fixed.
    - **Constraints**: ABI types are enforced, and each enabled direction requires a nonzero limit and window while disabled fields are unrestricted.
    - **Impact**: Each element checkpoints existing usage under its old per-EID or default config, stores the new config, and duplicates leave the last config effective.

Branches and code coverage

Intended branches

- Authorized empty batch: an empty owner batch succeeds without writes or events.
    - ☐ Test coverage
- Authorized nonempty batch: elements are processed sequentially in calldata order.
    - ☐ Test coverage
- Old effective configuration is a per-EID override: a preexisting override supplies the checkpoint limits and windows.
    - ☐ Test coverage

- Old effective configuration is the default: without a preexisting override, the checkpoint uses `defaultConfig`.
  - ☐ Test coverage
- Uninitialized or nonincreasing checkpoint time: usage is retained, and `lastUpdated` becomes the narrowed current timestamp.
  - ☐ Test coverage
- No-decay input: zero usage, limit, window, or elapsed time preserves usage while still checkpointing.
  - ☐ Test coverage
- Full-window refill: elapsed time at least equal to the old window reduces that direction's usage to zero.
  - ☐ Test coverage
- Partial-window refill: usage decreases by floor-rounded `oldLimit * elapsed / oldWindow`, bounded at zero.
  - ☐ Test coverage
- Enabled-direction validation succeeds: enabled directions require nonzero limits and windows, while disabled parameters may be zero or nonzero.
  - ☐ Test coverage
- Repeated EID: duplicates use the prior occurrence's config, preserve same-timestamp usage, and leave the last config stored.
  - ☐ Test coverage
- Stored override disabled: a supplied non-override config is stored and emitted, but later effective reads use the default.
  - ☐ Test coverage
- Rate limiting globally or directionally disabled: checkpointing still uses old limits and windows because enabled and global-disable flags are ignored.
  - ☐ Test coverage
- New limit below accrued usage: the config is accepted, but later availability remains zero until usage decays below the limit.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: the `onlyOwner` modifier reverts before the helper or library executes.
  - ☐ Negative test
- Nonzero native value: the nonpayable entry point rejects value before body execution.
  - ☐ Negative test

- Malformed calldata or out-of-range ABI value: ABI decoding rejects malformed tuples, noncanonical booleans, and values outside declared widths.
  - ☐ Negative test
- Invalid outbound configuration: an enabled outbound direction with a zero limit or window reverts before checkpointing that element.
  - ☐ Negative test
- Invalid inbound configuration: an enabled inbound direction with a zero limit or window reverts before checkpointing that element.
  - ☐ Negative test
- Linked-library failure or resource exhaustion: a library failure, revert, or out-of-gas condition rolls back all writes and logs.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
  - The generated `this` rendering is inaccurate. The source makes an internal helper call followed by a `DELEGATECALL` to a fixed library. The delegatecall preserves the caller, proxy storage, and zero value. It returns nothing, makes no callbacks or asset movements, and reverts all effects on failure.
- **Manual call-path verification**: The path is external `FraxOFTUpgradeable.setRateLimitConfigs` → `onlyOwner` → internal `RateLimiterModule._setRateLimitConfigs` → fixed `RateLimiterLib.setRateLimitConfigs`, then validation, old-config selection, checkpointing, state/config writes, and events.

Findings

- **Applicable per-EID behavior.** Zero limits in disabled directions preserve usage while advancing time, whereas nonzero placeholder limits decay usage during disabled intervals. Owner-selected fields determine usage history when the direction is reenabled.
- **Shared state behavior.** Both directions share a timestamp and use floor-rounded decay. Each EID's shared directional bucket permits contention among users but creates no authorization bypass or new issue here.

Function: `setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)`

After the ownership check succeeds, a fixed linked-library delegatecall stores the namespaced bool and emits `RateLimitGlobalConfigSet`. Setting `true` makes `quoteOFT` availability unlimited and bypasses `_debit` and `_credit` consumption, while setting `false` restores existing effective configurations without altering other limiter state.

Inputs

- `_globalConfig`
  - **Control**: The owner selects `_globalConfig.isGloballyDisabled`, while `msg.sender` gates ownership, `msg.value` must be zero, and `tx.origin` is unused.
  - **Constraints**: The canonical ABI bool is forwarded unchanged, malformed calldata reverts, and the linked-library address is deployment-fixed.
  - **Impact**: Success stores the bool and emits its matching event, while failure leaves storage, existing configurations, counters, and timestamps unchanged.

Branches and code coverage

Intended branches

- An authorized call with `isGloballyDisabled == false` stores `false` and emits `RateLimitGlobalConfigSet(false)`.
  - ☐ Test coverage
- An authorized call with `isGloballyDisabled == true` stores `true` and emits `RateLimitGlobalConfigSet(true)`.
  - ☐ Test coverage
- Both valid bool values follow the same storage-write-then-event path without checkpointing, clearing, decay, token movement, or nested calls.
  - ☐ Test coverage
- The reported `RateLimiterModuleTest` result is 6/6 passing but does not establish exact branch-level coverage for this setter.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts at `onlyOwner` before library execution, storage mutation, or event emission.
  - ☐ Negative test

- A call carrying nonzero `msg.value` reverts because the function is nonpayable.
  - ☐ Negative test
- Short, malformed, or noncanonical-bool calldata reverts during dispatch or decoding without state changes or events.
  - ☐ Negative test
- A linked-library failure propagates and atomically rolls back the write and log, although the resolved library accepts either valid bool.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The generated `this.` edge is incorrect because `_setRateLimitGlobalConfig` is internal and nonvirtual, while the terminal fixed-target `DELEGATECALL` is correct, preserves caller storage and sender, returns no value, and propagates failure without exposing callbacks or reentrancy.
- **Manual call-path verification**: The resolved path is `setRateLimitGlobalConfig` → `RateLimiterModule._setRateLimitGlobalConfig` → linked `RateLimiterLib.setRateLimitGlobalConfig` → namespaced write → event; unrelated ASI builder errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` affect other send or payment paths.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

This owner-only setter forwards raw per-EID states to a deployment-linked library that validates and writes the caller's fixed storage namespace. Selected values govern later availability, decay, checkpointing, and consumption.

Inputs

- `_params`
  - **Control**: The caller supplies ordered EIDs and raw outbound usage, inbound usage, and timestamps.
  - **Constraints**: Calldata must decode, timestamps cannot be in the future, and a zero timestamp requires both usages to be zero.
  - **Impact**: Each valid state overwrites its EID entry and emits once, with the last duplicate determining final storage.
- `msg.sender`

  - **Control**: The transaction caller determines `msg.sender`, which is preserved through proxy execution and the library delegatecall.
  - **Constraints**: The `onlyOwner` modifier requires `msg.sender` to be the current owner.
  - **Impact**: It controls authorization but is otherwise unused by the helper and library.
- Current ownership state
  - **Control**: The inherited ownership mechanism maintains the current owner in preexisting state.
  - **Constraints**: Authorization compares `msg.sender` with that owner.
  - **Impact**: It determines helper reachability and is not modified here.
- `msg.value`
  - **Control**: The caller selects transaction value.
  - **Constraints**: The nonpayable entry point rejects nonzero value.
  - **Impact**: Successful execution preserves zero value through `DELEGATECALL` without native-value movement.
- `block.timestamp`
  - **Control**: The current block supplies this environmental value.
  - **Constraints**: Every supplied `lastUpdated` must be no later than `block.timestamp`.
  - **Impact**: It validates timestamps, while the supplied timestamp itself is stored unchanged.
- `address(this)` and existing rate-limiter storage.
  - **Control**: The active proxy or implementation context determines `address(this)` and existing storage.
  - **Constraints**: The library writes only the fixed rate-limiter namespace keyed by each supplied EID.
  - **Impact**: Selected states are overwritten, and logs originate from the active calling address.
- Linked RateLimiterLib address and bytecode
  - **Control**: Deployment-time linking fixes the library target outside caller control.
  - **Constraints**: Analysis assumes linkage to the reviewed RateLimiterLib bytecode.
  - **Impact**: Its `DELEGATECALL` validates, writes, and emits in caller context without further external calls.
- `tx.origin`
  - **Control**: The originating account determines `tx.origin`.
  - **Constraints**: No constraint applies because this path never reads it.
  - **Impact**: It has no authorization, storage, event, or dispatch effect.

Branches and code coverage

Intended branches

- An authorized empty batch succeeds without writes or events.
  - ☐ Test coverage
- A zero timestamp with zero inbound and outbound usage is stored and emitted.
  - ☐ Test coverage
- A nonzero, nonfuture timestamp is accepted regardless of usage relative to configured limits.
  - ☐ Test coverage
- Multiple distinct EIDs are validated, written, and emitted in calldata order.
  - ☐ Test coverage
- Duplicate EIDs emit per occurrence, with the last valid occurrence winning.
  - ☐ Test coverage

Negative behavior

- A non-owner caller reverts before the helper or library executes.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point.
  - ☐ Negative test
- Malformed calldata or out-of-width encoded members revert during ABI decoding.
  - ☐ Negative test
- A future `lastUpdated` reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- A zero timestamp with either usage nonzero reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- Any later-element failure rolls back all earlier writes and logs in the batch.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
  - The `this.` prefix misrenders the internal nonvirtual helper call. The linked-library hop is a compiler-generated `DELEGATECALL` that preserves caller context, returns no value, and propagates reverts atomically.

- **Manual call-path verification**: The wrapper, inherited helper, and library resolve this path. Parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern unrelated `_quote` expressions and do not affect the trace.

5.9. Module: FraxOFTUpgradeableTempo.sol

Function: `constructor(address _lzEndpoint)`

This function deploys the implementation, forwards `_lzEndpoint` into OFT construction, and immutably stores the `nativeToken` returned by the endpoint. No caller authorization or local address validation exists.

Inputs

- `_lzEndpoint`
  - **Control**: The deployment account or factory supplies the endpoint without a local authorization check.
  - **Constraints**: The address is forwarded unchanged, and `nativeToken()` must return ABI-decodable address data. A zero or arbitrary decoded address is permitted.
  - **Impact**: The argument configures inherited OFT construction and the immutable token later used for whitelisting and fee-token wrapping.

Branches and code coverage

Intended branches

- Construction and inherited setup complete against the tested alternate endpoint, and deployed proxy runtime handles fee payment and cross-chain delivery.
  - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_PayNative_NoSwap_PathUsd`, `FraxOFTUpgradeableTempoTest.test_AltEndpoint_CrossChain_SendReceive_PathUsdNative`)
- The supplied endpoint is used by both inherited OFT operation and alternate-native-token payment in the tested deployment.
  - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_CrossChain_SendReceive_Pa`
- An ABI-compatible endpoint may return zero or an arbitrary address because the decoded result is not locally validated.
  - ☐ Test coverage — no supplied test is mapped specifically to zero or arbitrary

ABI-valid `nativeToken()` return data.

- The supplied constructor bodies have no conditional or loop, so outcomes depend on construction and external-call success.

  - ☐ Test coverage — this is exact-source review evidence rather than mapped branch-test evidence.

Negative behavior

- Nonzero `msg.value` reverts because the constructor is nonpayable.

  - ☐ Negative test — no exact constructor-value test was supplied.
- Any inherited-constructor revert atomically prevents deployment.

  - ☐ Negative test — no exact inherited-constructor failure test was supplied.
- A revert, out-of-gas failure, or ABI-invalid return from the `nativeToken()` query atomically prevents deployment.

  - ☐ Negative test — no exact endpoint-query failure test was supplied.
- An unintended token address returned with valid ABI encoding is accepted and may break later quote or fee-payment paths.

  - ☐ Negative test — no exact incorrect-endpoint or unintended-returned-token test was supplied.

Function call analysis

- `FraxOFTUpgradeable(_lzEndpoint)`

  - This fixed base-construction edge forwards the deployment-controlled endpoint and reverts deployment on failure.
- `OFTUpgradeable(_lzEndpoint)`

  - The imported constructor receives the endpoint unchanged, but its internal validation and interactions are unavailable in the supplied source.
- `_disableInitializers()` in FraxOFTUpgradeable

  - This fixed internal framework call occurs before the Tempo endpoint query and must complete for construction to continue.
- `TempoAltTokenBase(_lzEndpoint)`

  - This fixed base constructor queries the deployment-selected endpoint and assigns the decoded result to immutable `nativeToken`.
- `IEndpointV2Alt(_lzEndpoint).nativeToken()`

  - This zero-value `STATICCALL` decodes the address returned by the deployer-selected endpoint. Static context prevents state changes, and the unfinished contract has no runtime code to reenter.
- `_disableInitializers()` in FraxOFTUpgradeableTempo

    - This second fixed internal call executes after base construction and must complete before runtime code is deployed.
- **Manual call-path verification**: No FCA sink header was supplied. Exact-source tracing resolves `FraxOFTUpgradeableTempo.constructor` → FraxOFTUpgradeable/OFTUpgradeable construction → first `_disableInitializers()` → TempoAltTokenBase → `nativeToken()` → immutable assignment → final `_disableInitializers()`. The ASI failures at `FraxOFTUpgradeableTempo.sol@48:2180` and `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` are unrelated.

Findings

- No constructor-specific vulnerability was identified, but deployment must supply the intended trusted endpoint because neither it nor its returned token is locally authenticated.
- **EIP-712 initialization.** Neither `__EIP712_init` helper is called. ABI inspection found only guarded external initialization wrappers, while imported framework internals remain unresolved.

Function: `send(SendParam _sendParam, MessagingFee _fee, address _refundAddress)`

Permissionless sends reject nonzero `msg.value`, optionally apply rate limits, burn principal, build and inspect packets, pay fees, submit through LayerZero, return receipts, and emit `OFTSent`. The fee token and input cap remain unbound. Zero-equivalent EVM recipients may pass, refund asset and recipient paths differ, shared per-EID capacity permits contention, configuration changes use shared floored recovery, and `_toSD` may truncate principal. No freeze or pause hook exists.

Inputs

- `_sendParam`
    - **Control**: The caller controls the fields, while privileged state controls peers, options, inspector, and limits.
    - **Constraints**: Dust-cleaned amounts must satisfy slippage, capacity, balance, and encoding checks. The `_toSD` function has no shared-decimal ceiling, so values above `uint64` truncate.
    - **Impact**: It determines routing, principal, options, composition, rate usage, burn, and receipts.
- `_fee`
    - **Control**: The caller controls the amounts, infrastructure selects the tokens, and the owner controls swap slippage.
    - **Constraints**: Positive fees require successful resolution, transfers,

approvals, swaps, or wrapping.
    - **Impact**: It funds fees, and `uint128` narrowing may consume prefunded target tokens.
- `_refundAddress`
    - **Control**: The caller supplies this address without ownership or signature authorization.
    - **Constraints**: No local nonzero check applies; the endpoint may reject it.
    - **Impact**: It receives endpoint-dollar excess, while swap-input excess returns to `msg.sender`.

Branches and code coverage

Intended branches

- Zero `msg.value` proceeds without forwarding EVM-native value.
    - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns`
- Dust-cleaned zero proceeds without rate-limit storage changes.
    - ☐ Test coverage
- Positive dust-cleaned principal satisfying slippage drives accounting and burn.
    - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns` `FraxOFTUpgradeableTempoTest.test_AltEndpoint_CrossChain_SendReceive_PathUsdNative`)
- With global and outbound limiting disabled by default, the tested send succeeds without a rate-limit rejection.
    - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns`
- The disabled path skips usage changes and `RateLimitConsumed`.
    - ☐ Test coverage
- Enabled capacity updates shared usage and emits `RateLimitConsumed`.
    - ☐ Test coverage
- Successful burn reduces sender balance and supply without freeze checks.
    - ☐ Test coverage
- Nonzero `bytes32 to` accepts recipients that are equivalent to the zero address when interpreted as EVM addresses.
    - ☐ Test coverage
- An empty `composeMsg` completes a send and returns valid messaging and OFT receipts.

☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns`)

- Packet-type selection is not directly inspected.

☐ Test coverage

- Nonempty `composeMsg` embeds `msg.sender` for asynchronous send-and-call.

☐ Test coverage

- Without enforced options, caller `extraOptions` pass through unchanged.

☐ Test coverage

- Enforced options accept valid Type-3 options and concatenate them.

☐ Test coverage

- A zero inspector skips packet inspection.

☐ Test coverage

- A configured inspector statically validates messages and options.

☐ Test coverage

- Zero `nativeFee` skips fee-token resolution and movement.

☐ Test coverage

- A whitelisted token is transferred, approved, wrapped, and sent.

☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_PayNative_NoSwap_PathUsd`)

- The tested nonwhitelisted token follows a viable bounded DEX route and wraps the resulting pathUSD for the endpoint.

☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_PayNative_SwapToPathUsd`)

- Cheapest-route selection among multiple viable targets is not isolated.

☐ Test coverage

- Swap input below the cap resets allowance and refunds `msg.sender`.

☐ Test coverage

- Swap input at or above the cap skips reset and refund.

☐ Test coverage

- A send with zero `lzTokenFee` completes without a supplied LayerZero-token fee and returns valid receipts.

☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns`)

- The internal `payInLzToken` flag is not directly inspected.

☐ Test coverage

- A positive `lzTokenFee` transfers the endpoint token and sets `payInLzToken`.
  - ☐ Test coverage
- A configured peer permits endpoint submission and a nonzero messaging GUID.
  - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns`
- The endpoint receiver argument is not directly asserted.
  - ☐ Test coverage
- Successful submission returns the asserted messaging and OFT receipts.
  - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_Send_SucceedsWhenMsgValueZero_Returns`
- Successful submission emits `OFTSent`.
  - ☐ Test coverage
- Endpoint-dollar excess is refunded to `_refundAddress`.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` reverts with the alternate-endpoint value error.
  - ☑ Negative test (`FraxOFTUpgradeableTempoTest.test_Send_RevertsWhenMsgValueNonZero`)
- Precedence before debit, calls, and logs is not independently asserted.
  - ☐ Negative test
- Amounts below `minAmountLD` revert before capacity consumption.
  - ☐ Negative test
- Insufficient capacity reverts without retaining checkpoints, events, or burns.
  - ☐ Negative test
- Insufficient balance reverts and rolls back rate-limit effects.
  - ☐ Negative test
- Canonical zero recipients revert, while nonzero `bytes32` recipients that are equivalent to the zero address when interpreted as EVM addresses remain accepted.
  - ☐ Negative test
- Invalid Type-3 options revert when enforced options exist.
  - ☐ Negative test
- Inspector failure reverts the complete transaction.
  - ☐ Negative test
- Native-fee payment failures revert the complete transaction.

- ☐ Negative test
- Above-`uint128` swaps truncate; prefunding may still satisfy the full wrap.
  - ☐ Negative test
- LZ-token lookup, decoding, or transfer failure reverts everything.
  - ☐ Negative test
- A missing peer reverts and atomically undoes prior fee attempts.
  - ☐ Negative test
- Endpoint rejection or malformed receipts roll back all transactional effects.
  - ☐ Negative test

Function call analysis

- ASI trace: `_debit -> _consumeOutboundRateLimit -> RateLimiterLib.consumeRateLimit`.
- ASI trace: `_buildMsgAndOptions -> OFTMsgCodec.encode -> combineOptions -> IOAppMsgInspector.inspect`.
- ASI trace: `_lzSend -> _payLzToken -> endpoint.lzToken -> SafeERC20.safeTransferFrom`.
- ASI trace: `_lzSend -> endpoint.send -> _getPeerOrRevert`.
- ASI trace: `_lzSend -> _payNative -> _payNativeAltToken -> TempoAltTokenLib.payNativeAltToken`.
  - Inspector calls use `STATICCALL`, while library calls use `DELEGATECALL`; token, DEX, fee-manager, and endpoint-callback effects remain atomic. The fee token and input cap are unbound. Refund asset and recipient paths differ, shared per-EID capacity permits contention, configuration transitions use shared floored recovery, and `_toSD` may truncate principal.
- **Manual call-path verification**: Source review confirms that the generated trace omits the burn, compose sender, events, slippage read, and Tempo fee path. The generated `this.*` dispatch is inaccurate, and parser failures affect only scoped `_quote` overrides.

Function: `setFeeSwapSlippageBps(uint16 _bps)`

This owner-only wrapper forwards `_bps` unchanged to the inherited setter, which caps it at 200 and stores and emits the raw `uint16`. The getter maps zero to the 50-bps default. The effective value affects only nonwhitelisted fee-token swaps, and the fixed `send` function still exposes no caller-selected fee token or prior-quote maximum.

Inputs

- `_bps`

- **Control**: Calldata controls `_bps`, but `onlyOwner` restricts write authority to the current owner.
- **Constraints**: Values 0–200 are accepted, 201–65,535 revert, and nonzero `msg.value` is rejected before authorization.
- **Impact**: Success stores the raw value at `0x8c8147e168837ec687c20ac8f558b015dbf33e0ef9d2d8c77772e8c47e12ff00` and emits it, while later nonwhitelisted swaps use the effective value for quote padding, maximum input, and refunds.

Branches and code coverage

Intended branches

- With `_bps == 0`, the owner stores and emits zero while subsequent getter reads return 50 bps.
  - ☐ Test coverage — the supplied `TempoFeeSwapSlippageTest` summary reports 9/9 passing and bounded-bps coverage, but no exact named test and assertions were supplied for this target entry point's zero-storage, zero-event, and effective-default behavior.
- With `1 <= _bps <= 200`, the owner stores and emits the exact value, which the getter returns unchanged.
  - ☐ Test coverage — the suite summary reports bounded-bps coverage but does not provide an exact named test and assertions mapped to this target wrapper and accepted range.
- Later nonwhitelisted quote or payment paths apply the effective setting to the padded or executable maximum input.
  - ☐ Test coverage — the summary reports minimum one-unit headroom, quote/fill divergence, refund, and overpayment-conservation coverage, but no exact test source was supplied to map those assertions through this target's configured storage value.
- Later directly whitelisted payments transfer and wrap the native fee without using this setting.
  - ☐ Test coverage — no exact mapped test was supplied for this setting's non-effect on the direct branch.

Negative behavior

- A non-owner call reverts before the setter executes, leaving storage and logs unchanged.
  - ☐ Negative test — no exact unauthorized-caller test mapped to this target declaration was supplied.
- A value above 200 reverts with `FeeSwapSlippageTooHigh(_bps)` before writing or emitting.

- ☐ Negative test — bounded-bps coverage is reported only at suite-summary level, without an exact named test and assertions for this target's revert branch.

- A call with nonzero `msg.value` is rejected by nonpayable dispatch before authorization or execution.
  - ☐ Negative test — no exact nonpayable-call test was supplied.
- Any transaction-level failure rolls back the write and event, with execution ordered as authorization, cap check, write, then emission.
  - ☐ Negative test — no exact rollback test was supplied.

Function call analysis

- `this._setFeeSwapSlippageBps(_bps) -> this._getAltTokenStorage()`
  - These are internal calls mislabeled as `this.*`. The Yul helper only assigns the fixed storage slot and performs no external interaction.
- **Manual call-path verification**: Nonpayable dispatch and `onlyOwner` precede the inherited 200-bps cap, fixed-slot write, and event. No external call or reentrancy point occurs, and the ASI parser failure at `FraxOFTUpgradeableTempo.sol@48:2180` belongs to sibling `_quote` and is unrelated.

5.10. Module: FraxswapRouterMultihopV2.sol

Function: `constructor(IWETH _WETH9)`

This function stores `_WETH9` immutably and creates one fixed FraxswapExecutorV2 with the same dependency. No owner or admin is initialized; the executor records the constructing router as `router`, preventing direct deployer `execute` calls.

Inputs

- `_WETH9`
  - **Control**: The deployment context selects `_WETH9`, which is forwarded unchanged to the fixed executor creation.
  - **Constraints**: Neither constructor validates zero addresses, deployed code, interface support, trusted identity, or WETH semantics.
  - **Impact**: Both contracts store the value immutably, so invalid or malicious configuration permanently affects later wrapping, transfers, balance checks, and withdrawals.

Branches and code coverage

Intended branches

- Router construction stores a usable WETH dependency and creates the executor returned by `executor()`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_RevertWhen_NotRouter`)
- Construction causes no ETH or token movement.
  - ☐ Test coverage
- The created executor recognizes only its deploying router and uses the same WETH dependency during routed swaps.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_RevertWhen_NotRouter`)
- Transient state is untouched: neither constructor accesses router `locked`, executor `callbackAuth`, or sibling plug-in transient state.
  - ☐ Test coverage

Negative behavior

- Nonpayable creation: nonzero router creation value reverts at the generated nonpayable boundary, and executor creation receives zero ETH.
  - ☐ Negative test
- Executor creation failure: any `CREATE` failure reverts deployment atomically, leaving neither contract nor immutable assignment observable.
  - ☐ Negative test
- Invalid WETH is accepted: zero, EOA, non-WETH, or malicious `_WETH9` values are accepted and defer failure or malicious behavior until later use.
  - ☐ Negative test
- Operational environment without EIP-1153: construction executes no `TLOAD` or `TSTORE`, but guarded router and UniV3/UniV4 execution requires an EIP-1153–capable environment.
  - ☐ Negative test

Function call analysis

- `FraxswapRouterMultihopV2.constructor(_WETH9) -> new FraxswapExecutorV2(_WETH9) -> FraxswapExecutorV2.constructor(_WETH9)`
  - The sole interaction is a fixed `CREATE` with zero ETH. Successful creation supplies the immutable `executor` address, while failure bubbles and rolls

back. The executor constructor only assigns immutables, without events, callbacks, or external calls.

- **Manual call-path verification**: The artifact lacked an FCA sink header, but source review confirms this sole `CREATE` path and no constructor transient access; exact ASI parsing failed on four EIP-1153 `transient` declarations in failure.json, while transient-stripped generation reported no further parser or builder error.

Function: `execute(bytes route, uint256 amountIn, address tokenOut, address payable recipient, uint256 amountOutMinimum)`

The payable `FraxswapExecutorV2.execute` function accepts only its deploying router, executes user-controlled routes, checks full output balances, and pays `recipient`. ERC-20 fees may reduce receipt.

Inputs

- `route`
  - **Control**: The user-controlled route selects steps, targets, funding, callbacks, receivers, and accounting.
  - **Constraints**: Parsing, arithmetic, decoding, indexes, token matching, and final cursor checks may revert.
  - **Impact**: Routes can consume ambient balances or leave allowances, positions, and outputs in the executor for later public executions.
- `amountIn`
  - **Control**: The user declares `amountIn`, while the router selects its funding mechanism.
  - **Constraints**: Seeded credit is not reconciled against receipt, consumption, value, or refunds.
  - **Impact**: Underspend or repeated authenticated payments can consume or leave shared balances.
- `tokenOut`
  - **Control**: The user selects native output or any ERC-20 token.
  - **Constraints**: ERC-20 mode requires successful `balanceOf` decoding and an accepted transfer.
  - **Impact**: Payout includes the executor's entire measured balance, including ambient residue.
- `recipient`
  - **Control**: The user selects the recipient; the router resolves zero to its caller.
  - **Constraints**: Native calls and ERC-20 transfers must succeed.
  - **Impact**: Router recipients can trap payouts; executor recipients can leave

shared residue available to later public executions.

- `amountOutMinimum`
  - **Control**: The user supplies the nonzero minimum output floor.
  - **Constraints**: The full measured output balance must meet the floor.
  - **Impact**: It limits route underperformance but not transfer fees or per-step accounting.

Branches and code coverage

Intended branches

- The `onlyRouter` check accepts only the immutable deploying router before route execution.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_RevertWhen_NotRouter`)
- Asserted routes fund the executor with native input and ordinary ERC-20 input.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_ETH_FRAX`, `FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`)
- WETH funded by native value and direct-first-target funding lack explicit assertions.
  - ☐ Test coverage
- Tested DAG and three-hop routes consume the initial amount and pass credited intermediate output into later swaps.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`, `FraxswapRouterMultihopV2Test.test_swap_ThreeHop_DirectChain`)
- Direct assertions of allocation size and the initial `credits[0]` write are absent.
  - ☐ Test coverage
- Tested steps consume tracked credit or measured executor balance and dispatch positive amounts.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`, `FraxswapRouterMultihopV2Test.test_swap_FeeOnTransfer_MeasuredBasis`)
- Zero computed amounts are skipped.
  - ☐ Test coverage
- Percentages of at least `10_000` consume remaining value; smaller values use original basis.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`,

FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS)

- Explicitly asserted routes dispatch v2, v3, generic-call, and v4 step types and produce final output.
  - ☑ Test coverage (FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS, FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH, FraxswapRouterMultihopV2Test.test_swap_DAI_USDC_LitePSM, FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4)
- Optional credit output is covered separately by the DAG assertions.
  - ☐ Test coverage
- Tested v2 routes select their encoded quoting, fee, funding, direction, and receiver modes and deliver output.
  - ☑ Test coverage (FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS, FraxswapRouterMultihopV2Test.test_swap_FRAX_Velodrome_Fraxtal)
- Untested combinations of v2 flags remain uncovered.
  - ☐ Test coverage
- An asserted v3 DAG route produces and consumes positive output credit, while an unarmed callback is rejected.
  - ☑ Test coverage (FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH, FraxswapRouterMulti-hopV2Test.test_RevertWhen_V3CallbackUnauthorized)
- Postswap authorization clearing is not independently probed.
  - ☐ Test coverage
- The asserted LitePSM generic call uses its encoded approval, amount patching, target call, and output derivation modes.
  - ☑ Test coverage (FraxswapRouterMultihopV2Test.test_swap_DAI_USDC_LitePSM)
- Other generic-call funding, value, patch, and return-decoding combinations lack explicit assertions.
  - ☐ Test coverage
- A router-dispatched Pancake plug-in route reports positive output, and the plug-in's direct partial-fill path authenticates its callback and refunds the asserted remainder.
  - ☑ Test coverage (PancakeV3PluginTest.test_swap_USDC_WETH_Pancake, PancakeV3PluginTest.test_swap_PartialFillRefundsTheRemainder)
- Partial-fill refund through the full executor route is not isolated.
  - ☐ Test coverage
- Tested v4 ERC-20 and native-input routes authenticate the manager, settle their deltas, decode positive output, and complete.

- ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)

- Invalid-currency validation remains untested.
  - ☐ Test coverage

- Native payout unwraps routed WETH ,and ERC-20 payout transfers the routed measured output to the recipient.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_ETH`, `FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`)

- Payout of preexisting ambient output balances is not isolated.
  - ☐ Test coverage

Negative behavior

- A non-router caller with decodable arguments reverts with `FSE:OnlyRouter`.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_NotRouter`)

- ABI-decoding precedence for malformed calldata remains untested.
  - ☐ Negative test

- Deadline, minimum, value, direct-funding, wrapping, or input-transfer failures revert ingress.
  - ☐ Negative test

- A trailing route byte reverts with `FSE:RouteLength`, and a percentage above `10_000` reverts with arithmetic panic `0x11`.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_RouteTrailingBytes`, `FraxswapRouterMultihopV2Test.test_RevertWhen_PercentOver100`)

- Postrevert token and allowance state are not independently asserted.
  - ☐ Negative test

- Output below the floor reverts with `FSE:IO`.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_MaliciousV3PoolDoublePays`

- Rollback of every prior route effect is not independently asserted.
  - ☐ Negative test

- Failed pool, manager, plug-in, token, WETH, generic, or payout calls revert.
  - ☐ Negative test

- An unarmed v3 or Pancake callback is rejected, and an in-flight Pancake operation rejects reentry.

- ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_V3CallbackUnauthorized`, `PancakeV3PluginTest.test_RevertWhen_CallbackCalledDirectly`, `PancakeV3PluginTest.test_RevertWhen_PoolReenters`)

- A malicious authenticated v3 pool can invoke the callback repeatedly; the output-floor check catches the tested overdraft.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_MaliciousV3PoolDoublePays`)
- Repeated capped Pancake callbacks are permitted, and v4 trusts authenticated managers without cumulative bounds.
  - ☐ Negative test
- Executor residue is retained and remains available to later measured-balance routes.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_swap_frxUSD_USDC_Custodian_Partial`, `FraxswapRouterMultihopV2Test.test_swap_FindersKeepers_Dust`)
- Router recipients can trap payouts.
  - ☐ Negative test

Function call analysis

- `this.runRoute(...) -> IERC20(tokenIn).balanceOf(address(this))`
- `this.runRoute(...) -> this.executeStep(...) -> this.stepUniV2(...) -> pool quote/transfer/swap`
- `this.runRoute(...) -> this.executeStep(...) -> this.stepUniV3(...) -> pool.swap(...) -> authenticated callback`
- `this.runRoute(...) -> this.executeStep(...) -> this.stepCall(...) -> approval/transfer/WETH.withdraw/CALL/PancakeV3Plugin`
- `this.runRoute(...) -> this.executeStep(...) -> this.stepUniV4(...) -> manager.unlock(...) -> settlement callback`
- Native output uses `IERC20(WETH9).balanceOf(address(this)) -> WETH9.withdraw(...) -> recipient.call(...)`; ERC-20 output uses `IERC20(tokenOut).balanceOf(address(this)) -> TransferHelper.safeTransfer(...)`
- Route-selected external edges may return controlled data, retain allowances, repeat armed callbacks, or reduce transferred receipt.
- **Manual call-path verification**: Exact source includes router funding, approval probes, callbacks, settlement, and payout hooks; locking and `onlyRouter` block route-entry reentrancy, while authenticated targets may repeat callbacks. Four rejected `transient` declarations required stripping for generation; the generated output separately misrendered `this.*`, `new bytes[0]`, and the v3 price limit.

Function: `swap(FraxswapParams params)`

Permissionless exact-input routing uses a reentrancy lock, enforces deadlines, checks the aggregate output against the floor, and sweeps it.

Inputs

- `params`
  - **Control**: Any caller supplies the struct without authorization or allowlisting.
  - **Constraints**: Deadline, value, parsing, callback, call, cursor, and output checks apply.
  - **Impact**: It controls funding, execution, credits, output, recipient, return, and events.
  - `params.tokenIn`
    - **Control**: The caller selects native ETH, immutable WETH, or any ERC-20.
    - **Constraints**: Native and funded WETH require exact value; other inputs require zero.
    - **Impact**: It determines funding, credits, transfers, approvals, and callback payments.
  - `params.amountIn`
    - **Control**: The caller selects any unsigned nominal input, including zero.
    - **Constraints**: Balances, allowances, arithmetic, and v3/v4 `int256` bounds apply.
    - **Impact**: It determines funding, initial credit, route amounts, call value, and events.
  - `params.tokenOut`
    - **Control**: The caller selects native output or any ERC-20 contract.
    - **Constraints**: Balance queries, unwrapping, payment, and transfers must succeed.
    - **Impact**: It selects the executor balance to measure, check against the floor, and sweep.
  - `params.amountOutMinimum`
    - **Control**: The caller selects the output floor.
    - **Constraints**: It must be nonzero and not exceed measured output.
    - **Impact**: Failure reverts — ambient executor balances may satisfy the output floor without route-produced output.
  - `params.recipient`
    - **Control**: The caller selects any recipient; zero becomes

`msg.sender`.
- **Constraints**: Native calls and token transfers must succeed without address exclusions.
- **Impact**: ERC-20 output sent to the router becomes stranded; native payment to the router reverts.

- `params.deadline`
  - **Control**: The caller selects the timestamp deadline.
  - **Constraints**: The condition `block.timestamp <= params.deadline` must hold, including equality.
  - **Impact**: It bounds inclusion and is checked before funding.
- `params.route`
  - **Control**: The caller controls funding, venues, targets, hooks, credits, and payloads.
  - **Constraints**: Parsing, indexes, arithmetic, callbacks, calls, and cursor reconciliation must succeed.
  - **Impact**: It directs execution and may expose retained assets or stale approvals to later routes.

Branches and code coverage

Intended branches

- With `locked == 0`, hold the lock for the duration of successful execution.
  - ☐ Test coverage
- A deadline equal to `block.timestamp` is accepted.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`)
- A strictly later deadline is not isolated.
  - ☐ Test coverage
- Replace zero recipient with `msg.sender`; otherwise preserve it.
  - ☐ Test coverage
- Native input forwards exact `msg.value` through `executor.execute`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_ETH_FRAX`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- For WETH input funded with native value, deposit `msg.value` and transfer the WETH to `fundTarget(route)`.
  - ☐ Test coverage
- An ordinary ERC-20 input with zero call value is pulled and produces asserted output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`)

- With direct funding disabled, the tested route funds the executor and produces asserted output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`)
- Direct funding into `route[26:46]` lacks an explicit postcondition assertion.
  - ☐ Test coverage
- Each swap uses tracked credit or measured executor balance.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`, `FraxswapRouterMulti-hopV2Test.test_swap_FeeOnTransfer_MeasuredBasis`)
- Tested steps scale sub-100% amounts, consume the remaining credit at `10_000`, and deduct executed credit.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`, `FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`)
- A computed zero amount skips dispatch.
  - ☐ Test coverage
- Tested v2 flag combinations configure and complete Fraxswap and Velodrome swaps.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_swap_FRAX_Velodrome_Fraxtal`)
- Untested v2 flag combinations remain uncovered.
  - ☐ Test coverage
- V3 authorizes targets and callbacks and then derives output credit.
  - ☐ Test coverage
- The asserted LitePSM generic-call step patches its amount, uses its encoded approval mode, calls the target, and derives output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_DAI_USDC_LitePSM`)
- Other generic-call flag combinations lack explicit assertions.
  - ☐ Test coverage
- Tested v4 ERC-20 and native-input steps authorize the pool manager, settle, and decode positive output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- Invalid-currency and unauthorized-manager branches remain untested.
  - ☐ Test coverage

- A `creditTo` value of `0xFF` discards credit; otherwise, the `creditTo` slot increases.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`)
- Parsing requires the final cursor to equal `route.length`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_RevertWhen_RouteTrailingBytes`)
- Native output unwraps routed WETH, satisfies the output floor, and pays the measured swap output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_ETH`, `FraxswapRouterMultihopV2Test.test_swap_DAG_FRAX_ETH`)
- Sweeping a preexisting ambient ETH/WETH balance is not isolated.
  - ☐ Test coverage
- ERC-20 output satisfies the floor and transfers the measured routed token balance.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_swap_frxUSD_USDC_Custodian`)
- Sweeping a preexisting ambient output-token balance is not isolated.
  - ☐ Test coverage
- Success returns the measured `amountOut`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_FRAX_FXS`, `FraxswapRouterMultihopV2Test.test_swap_FRAX_ETH`)
- Success emits `Routed` and clears the lock.
  - ☐ Test coverage

Negative behavior

- Locked reentry into either swap entry point reverts with `FSR:Reentrancy`.
  - ☐ Negative test
- Expired deadlines revert with `Transaction too old` before funding.
  - ☐ Negative test
- A zero output floor reverts with `FSR:MZ`.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_MinimumOutZero`)
- An incorrect native/WETH value reverts with `FSR:II`; an unexpected value reverts with `FSR:IE`.
  - ☐ Negative test
- Malformed routes revert despite zero-padded assembly reads.
  - ☑ Negative test

(FraxswapRouterMultihopV2Test.test_RevertWhen_RouteTrailingBytes)

- Unknown steps revert with FSE:StepType; malformed values may create unusable credits.

  ☐ Negative test

- Invalid generic bounds or value tokens revert; target revert data bubbles.

  ☐ Negative test

- An unauthorized v3 callback reverts with FSE:V3Auth.

  ☑ Negative test (FraxswapRouterMultihopV2Test.test_RevertWhen_V3CallbackUnauthorized)

- Invalid v4 currency and nonpositive v3-delta failures remain untested.

  ☐ Negative test

- V3 lacks callback and cumulative caps, enabling authenticated pool overdrafts.

  ☑ Negative test (FraxswapRouterMultihopV2Test.test_RevertWhen_MaliciousV3PoolDoublePays

- V4 lacks outer-data binding and cumulative caps, enabling repeated altered callbacks.

  ☐ Negative test

- A generic route that tries to spend a victim's approval to the router reverts because the call executes from the executor, which has no such allowance.

  ☑ Negative test (FraxswapRouterMultihopV2Test.test_RevertWhen_RouteStealsApprovals)

- Other dependency, venue, approval, unwrap, and payout failures remain untested.

  ☐ Negative test

- If a malicious v3 pool leaves output below amountOutMinimum, the swap reverts with FSE:IO.

  ☑ Negative test (FraxswapRouterMultihopV2Test.test_RevertWhen_MaliciousV3PoolDoublePays

- Failed native or ERC-20 payout branches remain untested.

  ☐ Negative test

- Underallocation can retain executor residue, and a later measured-balance route can consume ambient executor funds.

  ☑ Negative test (FraxswapRouterMultihopV2Test.test_swap_frxUSD_USDC_Custodian_Partial, FraxswapRouterMultihopV2Test.test_swap_FindersKeepers_Dust)

- Stale approvals and internal-recipient trapping remain untested.

  ☐ Negative test

Function call analysis

- `this._swap(params) -> this.WETH9.deposit{value: msg.value}()`
- `this._swap(params) -> TransferHelper.safeTransfer(address(this.WETH9), this.fundTarget(params.route), msg.value)`
- `this._swap(params) -> TransferHelper.safeTransferFrom(params.tokenIn, msg.sender, this.fundTarget(params.route), params.amountIn)`
- `this._swap(params) -> this.executor.execute{value: value}(params.route, params.amountIn, params.tokenOut, address payable(recipient), params.amountOutMinimum)`
- External calls fund assets and invoke executor-selected targets; fee tokens may leave the credited amount underfunded.
- **Manual call-path verification**: Despite the `this.*` headers, `_swap` and `fundTarget` are internal. No `delegatecall` is involved. Ambient balances may satisfy the floor, router-directed ERC-20 output becomes stranded, and v3/v4 callbacks remain uncapped. ASI rejected four EIP-1153 `transient` declarations.

Function: `swapWithPermit(FraxswapParams params, bool approveMax, uint8 v, byte[32] r, byte[32] s)`

This payable exact-input swap attempts a permit, executes untrusted routes, pays the measured output, emits `Routed`, and returns the output.

Inputs

- `params`
  - **Control**: Caller selects all swap parameters.
  - **Constraints**: Guards, funding, routing, floor, and payout must succeed.
  - **Impact**: Determines execution, output, recipient, return, and event.
  - `params.tokenIn`
    - **Control**: Caller selects ETH, immutable WETH, or any ERC-20.
    - **Constraints**: Value and transfer requirements depend on funding.
    - **Impact**: Determines permit and funding behavior independently of the route tokens.
  - `params.amountIn`
    - **Control**: Caller selects the input amount and the permit value when `approveMax` is false.
    - **Constraints**: Zero is allowed if funding succeeds.
    - **Impact**: Sets funding, executor credit, and emitted input; it is the permit value only when `approveMax` is false.
  - `params.tokenOut`

    - **Control**: Caller selects ETH or any ERC-20 output.
    - **Constraints**: Balance measurement and payout must succeed.
    - **Impact**: Measured output may include ambient executor balances.
  - `params.amountOutMinimum`
    - **Control**: Caller selects the output floor.
    - **Constraints**: It must be nonzero and attainable.
    - **Impact**: It excludes neither ambient balances nor transfer fees.
  - `params.recipient`
    - **Control**: Caller selects the recipient; zero means `msg.sender`.
    - **Constraints**: Payout to the resolved recipient must succeed.
    - **Impact**: Router destinations can retain payouts; executor destinations can expose shared residue to later swaps.
  - `params.deadline`
    - **Control**: Caller selects the router and permit deadline.
    - **Constraints**: Execution must be timely; tokens may add rules.
    - **Impact**: Bounds execution and passes unchanged to `permit`.
  - `params.route`
    - **Control**: Caller selects steps, targets, tokens, percentages, offsets, and calldata.
    - **Constraints**: Parsing, accounting, callbacks, calls, and payout must succeed.
    - **Impact**: Under-use, approvals, donations, and residue can leave executor assets or spending authority available to later swaps.
- `approveMax`
  - **Control**: Caller selects exact or maximum permit value.
  - **Constraints**: Only non-native inputs use it; failures are caught.
  - **Impact**: It changes only the permit allowance; route execution and payout remain unbound, while funding is capped at `params.amountIn`.
- `v`
  - **Control**: Caller supplies the `uint8` signature component.
  - **Constraints**: The router does not validate it; the caller-selected permit target determines handling and may be nonconforming.
  - **Impact**: It is passed only to `permit`.
- `r`
  - **Control**: Caller supplies the 32-byte signature component.
  - **Constraints**: Source uses `bytes32` — the router does not validate it, and target failures are caught.

    - **Impact**: It is passed only to `permit`.
- `s`
    - **Control**: Caller supplies the 32-byte signature component.
    - **Constraints**: Source uses `bytes32`; the router does not enforce canonicality, and the caller-selected target controls acceptance.
    - **Impact**: It is passed only to `permit`.

Branches and code coverage

Intended branches

- Unlocked calls set and later clear the lock.
    - ☐ Test coverage
- Deadlines equal to `block.timestamp` remain valid.
    - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- Native input skips permit.
    - ☐ Test coverage
- Non-native input permits `params.amountIn` without `approveMax`.
    - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- Non-native input permits `type(uint256).max` with `approveMax`.
    - ☐ Test coverage
- Successful permit continues execution.
    - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- Reverting permit is caught for alternative funding.
    - ☑ Test coverage (`test_swap_Permit_Frontrun`)
- Nonzero output floors continue.
    - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- The tested nonzero recipient is preserved and receives the output.
    - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- A zero recipient aliases `msg.sender`.
    - ☐ Test coverage
- Native input forwards exactly `params.amountIn`.
    - ☐ Test coverage
- For WETH input funded with native value, the router deposits and transfers WETH then calls the executor with zero ETH.

☐ Test coverage

- ERC-20 funding pulls `params.amountIn` without value.

☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)

- Clear direct-fund bits select the executor.

☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)

- Set direct-fund bits select route-encoded targets.

☐ Test coverage

- The tested permit route uses its initial tracked credit.

☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)

- Measured executor-balance input in `swapWithPermit` remains untested.

☐ Test coverage

- The tested `10_000` percentage consumes the remaining credit.

☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)

- A below-`10_000` percentage in `swapWithPermit` remains untested.

☐ Test coverage

- Zero-step swaps may sweep ambient output.

☐ Test coverage

- Positive type-0 steps execute v2 flow.

☐ Test coverage

- Positive type-1 steps execute authenticated v3 callbacks.

☐ Test coverage

- Positive type-2 steps fund, patch, call, and decode.

☐ Test coverage

- The `PancakeV3Plugin.swap` function authenticates callbacks without cumulative callback payment caps.

☐ Test coverage

- The `PancakeV3Plugin.getAmountOut` function accepts any exactly 32-byte quote-revert payload from the caller-selected pool.

☐ Test coverage

- Positive type-3 steps execute authenticated v4 unlock.

☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)

- The tested `creditTo == 0xFF` discards intermediate credit.

☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)

- Increasing a credit slot in `swapWithPermit` remains untested.

- ☐ Test coverage
- Native output unwraps, measures, checks, and pays everything.
  - ☐ Test coverage
- ERC-20 output measures, checks, and transfers selected balance.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- Success returns the asserted ERC-20 output to the signer.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_Permit`)
- Success emits `Routed` and clears the lock.
  - ☐ Test coverage

Negative behavior

- Calls made while the lock is set revert with `FSR:Reentrancy`.
  - ☐ Negative test
- Expired deadlines revert with `Transaction too old`.
  - ☐ Negative test
- Caught permit failures may cause funding failure.
  - ☐ Negative test
- Zero floors revert with `FSR:MZ`.
  - ☐ Negative test
- Invalid value reverts with `FSR:II` or `FSR:IE`.
  - ☐ Negative test
- A decoded zero swap count panics. An individual swap with zero steps can succeed and sweep ambient output. Unsupported positive step types revert with `FSE:StepType`.
  - ☐ Negative test
- Malformed routes cause `FSE:RouteLength`, parsing, or arithmetic failures.
  - ☐ Negative test
- Uncaught external or payout failures revert atomically.
  - ☐ Negative test
- Generic failures bubble; invalid patches, return data, or values revert with `FSE` errors.
  - ☐ Negative test
- Invalid v3 callbacks trigger `FSE:V3Auth` or `FSE:V3Delta`; bounds remain absent.
  - ☐ Negative test
- Invalid v4 callers or tokens revert; callback bounds remain absent.
  - ☐ Negative test

- Pancake checks may revert; `PCV3:Overpay` remains noncumulative.
  - ☐ Negative test
- Output below the floor reverts with `FSE:IO`.
  - ☐ Negative test
- Failed ETH or ERC-20 payouts revert.
  - ☐ Negative test
- Router-directed token payouts remain trapped without a recovery path; router-directed ETH reverts.
  - ☐ Negative test
- Under-spend, approvals, donations, positions, and residue can leave executor assets or spending authority available to later swaps.
  - ☐ Negative test

Function call analysis

- `IERC20Permit(params.tokenIn).permit(msg.sender, address(this), amount, params.deadline, v, r, s)`
- `this._swap(params) -> this.WETH9.deposit{value: msg.value}()`
- `this._swap(params) -> TransferHelper.safeTransfer(address(this.WETH9), this.fundTarget(params.route), msg.value)`
- `this._swap(params) -> TransferHelper.safeTransferFrom(params.tokenIn, msg.sender, this.fundTarget(params.route), params.amountIn)`
- `this._swap(params) -> this.executor.execute{value: value}(params.route, params.amountIn, params.tokenOut, address payable(recipient), params.amountOutMinimum)`
- External calls reach untrusted contracts while the lock is set; v3/v4 callback bounds remain absent.
- **Manual call-path verification**: ASI misrendered internal `_swap` and `fundTarget`, omitted downstream calls and callbacks, stripped transient declarations, and showed `byte[32]` instead of source `bytes32`.

Function: `uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes _data)`

This function authenticates the caller/token commitment and transfers the selected positive delta. It writes no state or event, and authorization remains reusable during the pool call.

Inputs

- `amount0Delta`
  - **Control**: The authenticated callback caller supplies this delta.
  - **Constraints**: A positive value takes precedence and is not capped by the step's `amountIn`.
  - **Impact**: When positive, it determines the token amount transferred to `msg.sender`.
- `amount1Delta`
  - **Control**: The authenticated callback caller supplies this delta.
  - **Constraints**: It must be positive when `amount0Delta <= 0` and is otherwise ignored.
  - **Impact**: It determines the transfer amount only when `amount0Delta` is nonpositive.
- `_data`
  - **Control**: The pool controls the callback bytes despite receiving `abi.encode(tokenIn)` upstream.
  - **Constraints**: It must decode as an address matching the committed `(msg.sender, tokenIn)` pair.
  - **Impact**: The decoded address becomes the token call target and may be replayed while authorization remains live.
- `msg.sender` (implicit)
  - **Control**: The immediate external caller determines this address.
  - **Constraints**: It must match the target in the active transient commitment.
  - **Impact**: It is both the authenticated pool and token-transfer recipient.
- `msg.value` (implicit)
  - **Control**: The immediate caller selects the attached native value.
  - **Constraints**: The nonpayable entry check requires zero value.
  - **Impact**: It cannot fund or otherwise affect the token payment.
- `callbackAuth` (implicit transient state)
  - **Control**: The `stepUniV3` function sets it from the route-selected target and token.
  - **Constraints**: The callback checks equality but neither clears it nor tracks remaining allowance.
  - **Impact**: It authorizes transfers and remains reusable until the pool call returns.

Branches and code coverage

Intended branches

- Decode and authorization success: valid `_data` reproduces the live `(msg.sender, tokenIn)` commitment.
  - ☐ Test coverage
- Token0-delta payment: a positive `amount0Delta` is transferred and wins when both deltas are positive.
  - ☐ Test coverage
- Token1-delta payment: when `amount0Delta <= 0`, a positive `amount1Delta` is transferred.
  - ☐ Test coverage
- Accepted token-call return: empty return data or ABI-encoded `true` succeeds, including empty success from a no-code address.
  - ☐ Test coverage
- Upstream return: if the pool returns, the callback clears authorization and performs route-local output accounting.
  - ☐ Test coverage

Negative behavior

- Malformed callback data: invalid address encoding causes `abi.decode` to revert before any token call.
  - ☐ Negative test
- Unauthorized or out-of-window caller/token pair: a mismatched commitment reverts with `FSE:V3Auth`.
  - ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_V3CallbackUnauthorized`)
- No positive amount owed: nonpositive deltas revert with `FSE:V3Delta`.
  - ☐ Negative test
- Token call failure, false result, or malformed result: failed calls, `false`, or malformed nonempty boolean data revert.
  - ☐ Negative test
- Insufficient balance or token-specific rejection: token rejection propagates, while dishonest success is not balance-reconciled.
  - ☐ Negative test
- Nonzero native value: generated nonpayable entry code rejects the call before execution.
  - ☐ Negative test
- An authenticated malicious v3 pool can invoke the callback repeatedly and debit more than the nominal input when ambient executor funds exist; the final output floor then reverts the swap.

- ☑ Negative test (`FraxswapRouterMultihopV2Test.test_RevertWhen_MaliciousV3PoolDoublePays`)

- Reentrant request during token execution: authorization stays live under token control, although `onlyRouter` and the router lock block nested executor and router entry points.
  - ☐ Negative test
- Pool catches callback failure: a malicious pool may catch a reverted callback and return valid swap data, while reverted callback effects remain rolled back.
  - ☐ Negative test

Function call analysis

- `TransferHelper.safeTransfer(tokenIn, msg.sender, uint256(amount0Delta > 0 ? amount0Delta : amount1Delta))`
  - The inlined helper makes a low-level call to `tokenIn.transfer(msg.sender, selectedDelta)` without native value; token code controls effects and return data while authorization remains live.
- **Manual call-path verification**: The pool callback edge, reusable authorization, and settlement path were checked manually; the supplied cumulative-cap proof was not mapped to this executor selector; ASI failed on four EIP-1153 `transient` declarations, while the disclosed stripped workaround succeeded.

Function: `unlockCallback(bytes data)`

With transient `callbackAuth` matching `msg.sender`, this callback swaps caller data, settles unchecked deltas, requests output, and returns `abi.encode(amountOut)`. Payload and cumulative payment remain unbound.

Inputs

- `data`
  - **Control**: The authenticated caller controls amount, direction, pool key, hooks, and hook data.
  - **Constraints**: Authorization binds only `msg.sender`; 99 bytes and an `int256`-sized amount are required.
  - **Impact**: Caller-selected fields and deltas determine executor payment, requested output, and return value.

Branches and code coverage

Intended branches

- Matching transient authorization permits execution for the target committed by `stepUniV4`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- The tested nonzero direction selects native `currency0` input and `currency1` output and completes successfully.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- The exact lower price-limit constant is not asserted.
  - ☐ Test coverage
- The tested zero direction selects ERC-20 `currency1` input and `currency0` output and completes successfully.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`)
- The exact upper price-limit constant is not asserted.
  - ☐ Test coverage
- Both tested v4 directions decode pool deltas into the expected input settlement and positive output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- Raw signed-half values are not directly asserted.
  - ☐ Test coverage
- The selected input delta produces the amount successfully settled in each tested direction.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- Boundary behavior of the negation and `uint128` cast remains untested.
  - ☐ Test coverage
- Native input sends `owed` value through `settle`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- ERC-20 input performs `sync`, token transfer, and no-value `settle`.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`)
- The selected output delta becomes `amountOut`, which `take` requests.

- ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)

- The callback returns a decodable output amount that is credited and paid by each tested route.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- Independence from arbitrary upstream `unlock` return bytes is not isolated.
  - ☐ Test coverage
- Successful v4 unlocks produce final balance-checked output.
  - ☑ Test coverage (`FraxswapRouterMultihopV2Test.test_swap_USDT_USDC_UniV4`, `FraxswapRouterMultihopV2Test.test_swap_ETH_USDC_UniV4`)
- Authorization clearing before crediting is not independently probed.
  - ☐ Test coverage

Negative behavior

- Idle, unauthorized, or incorrect-caller invocation reverts with `FSE:V4Auth`.
  - ☐ Negative test
- Nonzero `msg.value` fails the nonpayable entry check.
  - ☐ Negative test
- Data shorter than 99 bytes reverts before `swap`.
  - ☐ Negative test
- Amounts above `type(int256).max` revert during `SafeCast.toInt256`.
  - ☐ Negative test
- Swap, malformed return, pool, or hook failure reverts the route.
  - ☐ Negative test
- Negating `type(int128).min` reverts; other deltas with an incorrect sign may become huge unsigned values.
  - ☐ Negative test
- Failed or underfunded native settlement reverts the route.
  - ☐ Negative test
- Failed ERC-20 synchronization, transfer, return handling, or settlement reverts.
  - ☐ Negative test
- Failed `take`, PoolManager credit, or output transfer reverts.

- ☐ Negative test

- The authenticated target may recursively repeat altered payloads without cumulative `owed` limits.
  - ☐ Negative test
- Malformed `unlock` bytes revert decoding; false credits remain balance-floor constrained.
  - ☐ Negative test
- A failed final output floor or payout reverts the route.
  - ☐ Negative test

Function call analysis

- `this.readU8(data, 32)`
- `this.readAddress(data, 33)`
- `this.readAddress(data, 53)`
- `IPoolManagerV4(msg.sender).swap(PoolKeyV4(currency0, currency1, uint24(this.readU24(data, 73)), int24(uint24(this.readU24(data, 76))), this.readAddress(data, 79)), SwapParamsV4(zeroForOne, -SafeCast.toInt256(this.readU256(data, 0)), zeroForOne ? 4295128740 : 1461446703485210175993845903335784348927318294528), data[99:])`
- `this.readU24(data, 73)`
- `this.readU24(data, 76)`
- `this.readAddress(data, 79)`
- `SafeCast.toInt256(this.readU256(data, 0))`
- `this.readU256(data, 0)`
- `IPoolManagerV4(msg.sender).settle{value: owed}()`
- `IPoolManagerV4(msg.sender).sync(currencyIn)`
- `TransferHelper.safeTransfer(currencyIn, msg.sender, owed)`
- `IPoolManagerV4(msg.sender).settle()`
- `IPoolManagerV4(msg.sender).take(zeroForOne ? currency1 : currency0, address(this), amountOut)`
- **Manual call-path verification**: Source confirms internal readers, source upper limit 1461446703485210103287273052203988822378723970341, `swap`, settlement, transfer, and `take`. Authorization remains armed for same-target recursion or repetition until `stepUniV4` clears it after a successful `unlock`.

Findings

- **Accepted public-executor policy.** Ambient executor balances may fund payments. Address-only authorization permits repeated callbacks without a cumulative cap. The

final measured-output floor is the bound.
- **ASI mismatches.** ASI uses the incorrect false-direction limit, renders internal readers as `this.*`, and rejects four EIP-1153 `transient` declarations. The function is nonvirtual and has no override. The sibling Pancake callback limit does not constrain this path.

5.11. Module: FreezeThawLib.sol

Function: `addFreezer(address account)`

Linked-library `DELEGATECALL` mutates the delegator's fixed-slot freezer set. Membership enables FrxUSD `freeze` and `freezeMany`, while the owner remains independently authorized. The target performs no caller authorization or address validation.

Inputs

- `account`
  - **Control**: The delegating contract supplies any ABI-decoded address unchanged to `EnumerableSet.add`, while the FrxUSD route restricts the external caller to `owner()`.
  - **Constraints**: Any valid address, including zero, succeeds when absent, while existing membership reverts with `FreezeThawModule.AlreadyFreezer()`.
  - **Impact**: Success stores and emits the address from the delegator, enabling FrxUSD freezer actions, while any failure rolls back all effects.

The delegator determines storage and `address(this)`. Delegatecall preserves unread `msg.sender` and `msg.value`, although FrxUSD's nonpayable `onlyOwner` wrapper rejects value before forwarding.

Branches and code coverage

Intended branches

- When `account` is absent, membership is stored in the delegating contract.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_AddFreezer_succeeds`)
- `FreezeThawModule.AddFreezer(account)` is emitted from the delegating contract.
  - ☐ Test coverage
- An absent `address(0)` is accepted, stored, and emitted because the target performs no address validation.
  - ☐ Test coverage
- A payable delegator may reach success with preserved nonzero `msg.value`; the analogous Solc 0.8.22 `RateLimiterLib.consumeRateLimit` harness passed 2/2

invocations.

- ☐ Test coverage

- The owner-gated FrxUSD route forwards the supplied account to linked-library storage, where membership becomes visible.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_AddFreezer_succeeds`)

Negative behavior

- Adding an existing freezer reverts with `AlreadyFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_AddFreezer_AlreadyFreezer_reverts`)
- Postrevert membership and log state are not independently asserted.
  - ☐ Negative test
- FrxUSD's nonpayable wrapper rejects nonzero `msg.value` before authorization, hook execution, or delegatecall.
  - ☐ Negative test
- Malformed low-level delegatecall calldata fails selector or ABI decoding before storage access.
  - ☐ Negative test
- An ordinary state-changing `CALL` to the deployed library is rejected by Solidity's library runtime delegatecall guard.
  - ☐ Negative test
- Any failure in the helper, set operation, or event emission propagates and rolls back the transaction.
  - ☐ Negative test
- A non-owner FrxUSD caller is rejected.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_AddFreezer_OnlyOwner`)
- The test does not instrument whether library delegation was reached.
  - ☐ Negative test

Function call analysis

- `this._getFreezeThawStorage()`
  - The generated `this.` is incorrect; `_getFreezeThawStorage()` is a private pure helper that only assigns the fixed storage slot and makes no external call.
- `EnumerableSet.add($.freezers, account)`

    - This internal storage operation inserts `account` or returns `false` for existing membership without calling `account`, transferring value, or permitting reentrancy.
- **Manual call-path verification**: Normal linked invocation uses `DELEGATECALL` and preserves the outer context and value without transferring new value. FrxUSD's owner-gated nonpayable wrapper supplies the audited route. Any failure reverts the transaction.

Function: `freeze(address account)`

This function adds any address to the frozen set in the caller's storage without performing authorization and emits `AccountFrozen` even if the address is already present. Because the linked library runs via `delegatecall`, it uses the caller's storage context, address, `msg.sender`, and `msg.value`.

Inputs

- `account`
    - **Control**: The immediate caller supplies the address unchanged.
    - **Constraints**: Any address is accepted, including zero and existing members, although supplied wrappers restrict the caller.
    - **Impact**: It becomes the set key and event argument; zero blocks non-owner minting and burning in the FrxUSD consumer while owner-operated paths bypass the check.
- Prior `$.frozen` membership (implicit state input)
    - **Control**: Earlier freeze and thaw operations in the calling storage context determine membership.
    - **Constraints**: Membership is either absent or present in the fixed `frozen` set.
    - **Impact**: Absent membership is inserted, while present membership remains unchanged and still produces an event.
- `msg.sender` (implicit execution input)
    - **Control**: Delegatecall preserves the original caller of the outer entry point.
    - **Constraints**: This function does not inspect it, while supplied wrappers require a freezer or owner.
    - **Impact**: It affects wrapper authorization but has no direct sink in this body.
- `msg.value` (implicit execution input)
    - **Control**: Delegatecall preserves the outer call value without transferring new value to the library.
    - **Constraints**: The body ignores preserved value, while supplied nonpayable wrappers reject it before delegation.
    - **Impact**: Any accepted value remains in the calling contract and does not

affect execution.

- `address(this)` and storage context (implicit execution inputs)
  - **Control**: Under `delegatecall`, both refer to the calling contract or proxy.
  - **Constraints**: The `_getFreezeThawStorage()` helper selects `0xf6192aebe04228448ed4fd2802d59c009f5324cdd6a110c65bd318373b6e1d00`.
  - **Impact**: Membership writes and logs belong to the calling context, not the deployed library.
- Other environmental values, including `tx.origin`, `block.timestamp`, `block.number`, `block.chainid`, and calldata beyond the decoded `account` (implicit inputs)
  - **Control**: These values are externally or protocol determined.
  - **Constraints**: The body, storage helper, and set operation do not read them.
  - **Impact**: They do not affect authorization, membership, event data, or control flow.

Branches and code coverage

Intended branches

- An absent tested account is inserted into frozen membership.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeAsFreezer_succeeds`, `FrxUSDOFTUpgradeableTest.test_FreezeAsOwner_succeeds`)
- Insertion emits `AccountFrozen(account)`.
  - ☐ Test coverage
- An existing `account` remains unchanged but still succeeds and emits.
  - ☐ Test coverage
- The zero address (`address(0)`) follows normal membership behavior and blocks minting and burning in the FrxUSD consumer.
  - ☐ Test coverage
- A freezer or owner passes the supplied single-account wrapper authorization and reaches this function.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeAsFreezer_succeeds`, `FrxUSDOFTUpgradeableTest.test_FreezeAsOwner_succeeds`)
- The `freezeMany` function processes entries in order. Each duplicate entry that reaches processing emits an event, while membership remains a single set entry.
  - ☐ Test coverage

Negative behavior

- A caller that is neither freezer nor owner reverts with `NotFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Freeze_NotFreezer_reverts`)
- Reversion before delegation is not instrumented.
  - ☐ Negative test
- A direct external `CALL` cannot execute the body because Solidity's library guard requires delegatecall context.
  - ☐ Negative test
- Supplied nonpayable wrappers reject nonzero value, but execution completed with preserved `msg.value == 1` in 2/2 linked-library probe paths compiled with Solc 0.8.22.
  - ☐ Negative test
- Any delegation, storage, gas, or emission failure rolls back membership, logs, prior batch iterations, and preceding same-transaction updates.
  - ☐ Negative test
- The function does not reject zero, so an authorized caller can block non-owner minting and burning in the FrxUSD consumer while owner-operated paths bypass the check.
  - ☐ Negative test

Function call analysis

- `this._getFreezeThawStorage()`
  - ASI's `this.` qualification is incorrect; this private pure helper only assigns the fixed storage namespace and performs no external call.
- `EnumerableSet.add($.frozen, account)`
  - This internal, callback-free operation inserts only absent members, returns an ignored boolean, and propagates any failure.
- **Manual call-path verification**: The supplied path authorizes in FrxUSDOFTUpgradeable and enters the linked library through `delegatecall`. It then calls the storage helper and `EnumerableSet.add`. ASI omitted the `delegatecall`; direct `CALL`, token movement, value forwarding, and reentrancy are absent.

Function: `freezeMany(address[] accounts)`

This function adds each `accounts` entry to the fixed-slot frozen set and emits `AccountFrozen` for every entry. Empty arrays do nothing, while duplicates repeat events without changing membership. The library target has no validation, authorization, batch limit, or return value; the supplied `FrxUSDOFTUpgradeable.freezeMany` wrapper permits only a configured freezer or the owner.

Inputs

- `accounts`
  - **Control**: The invoker controls array length, order, duplicates, and address keys, while the delegatecall context selects the affected storage and log address.
  - **Constraints**: Calldata must decode canonically, but zero, duplicate, existing, contract, and EOA addresses are accepted, with practical size limited only by calldata and gas.
  - **Impact**: Each entry is added to the fixed-slot set and emits once; freezing zero blocks non-owner mint and burn paths while owner-initiated operations bypass the hook.

Branches and code coverage

Intended branches

- An empty array succeeds without storage changes or events.
  - ☐ Test coverage
- A tested two-element array produces two frozen-set entries.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeManyAsFreezer_succeeds`, `FrxUSDOFTUpgradeableTest.test_FreezeManyAsOwner_succeeds`)
- Element order and duplicate behavior remain untested.
  - ☐ Test coverage
- Each absent tested address is inserted into frozen membership.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeManyAsFreezer_succeeds`, `FrxUSDOFTUpgradeableTest.test_FreezeManyAsOwner_succeeds`)
- Each insertion emits `AccountFrozen`.
  - ☐ Test coverage
- An existing or duplicate address remains a single member but emits again.
  - ☐ Test coverage
- The zero address (`address(0)`) is processed as an ordinary set key.
  - ☐ Test coverage

Negative behavior

- Malformed array encoding or a noncanonical address word fails decoding before execution.

- ☐ Negative test

- A linked-library `DELEGATECALL` may preserve nonzero value, although the supplied nonpayable wrapper rejects it beforehand and the target ignores it.
  - ☐ Negative test
- A direct `CALL` cannot execute this state-mutating deployed-library function.
  - ☐ Negative test
- The target does not reject unauthorized callers or any address category.
  - ☐ Negative test
- Oversized arrays can exhaust calldata or execution gas, with no partial success.
  - ☐ Negative test
- Static execution, revert, delegatecall failure, or out-of-gas rolls back all writes and logs atomically.
  - ☐ Negative test

Function call analysis

- `this.freeze(accounts[i]) -> this._getFreezeThawStorage()`
  - The generated `this.` labels are inaccurate; both calls are direct/internal, and the Yul getter only assigns the fixed `FreezeThawStorageLocation` without external interaction.
- `this.freeze(accounts[i]) -> EnumerableSet.add($.frozen, account)`
  - The `EnumerableSet.add` function performs internal storage bookkeeping, its membership boolean is discarded, and it exposes no callback or reentrancy edge.
- **Manual call-path verification**: The actual path is `FreezeThawLib.freezeMany -> FreezeThawLib.freeze -> FreezeThawLib._getFreezeThawStorage -> EnumerableSet.add -> emit AccountFrozen`; linked invocation uses one outer `DELEGATECALL`, failures propagate atomically, and unrelated parser failures in FraxOFTMintableAdapterUpgradeableTIP20.sol and FraxOFTUpgradeableTempo.sol do not affect this path.

Function: `removeFreezer(address account)`

This linked-library delegatecall removes `account` from shared namespace `0xf6192aebe04228448ed4fd2802d59c009f5324cdd6a110c65bd318373b6e1d00` and emits `FreezeThawModule.RemoveFreezer(account)` from the caller's address. The target lacks authorization and value checks; the intended `FrxUSDOFTUpgradeable.removeFreezer` wrapper enforces `onlyOwner` and nonpayability.

Inputs

- `account`
  - **Control**: The caller supplies the address, which the intended wrapper forwards unchanged.
  - **Constraints**: Any decoded address is accepted, including zero, but successful removal requires current set membership.
  - **Impact**: The address is removed from `$.freezers` and emitted unchanged in `FreezeThawModule.RemoveFreezer(account)`.
- `msg.sender` (implicit input)
  - **Control**: Delegatecall preserves the original caller selected through the outer call path.
  - **Constraints**: The target does not read or authenticate `msg.sender`, while the intended wrapper applies `onlyOwner`.
  - **Impact**: It does not affect storage selection, membership, events, or target control flow.
- `msg.value` (implicit input)
  - **Control**: Delegatecall preserves the outer call's value and cannot attach a new value.
  - **Constraints**: The target does not inspect value, although the intended nonpayable wrapper rejects nonzero value before entry.
  - **Impact**: No ETH is transferred or refunded, and preserved value remains in the calling contract.
- `tx.origin` (implicit input)
  - **Control**: The transaction originator determines `tx.origin`, which is preserved through the call path.
  - **Constraints**: Neither the target nor its internal callees reads it.
  - **Impact**: It has no effect on authorization, storage, membership, events, or control flow.

Branches and code coverage

Intended branches

- A present account is removed from the caller's namespaced storage.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_RemoveFreezer_succeeds`)
- Removal is logged from the caller's address.
  - ☐ Test coverage
- The zero address (`address(0)`) is removable and logged if it is currently a member.

- ☐ Test coverage

- A payable linking caller may preserve nonzero `msg.value` through Solc 0.8.22 delegatecall without affecting membership behavior.
  - ☐ Test coverage
- The owner-gated concrete wrapper reaches linked-library removal and the tested freezer membership is removed.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_RemoveFreezer_succeeds`)
- Invocation count is not independently instrumented.
  - ☐ Test coverage

Negative behavior

- Removing an absent account reverts with `NotFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_RemoveFreezer_NotFreezer_reverts`)
- Postrevert storage and log state are not independently asserted.
  - ☐ Negative test
- The target does not reject unauthorized callers, so authorization must be imposed by the calling integration.
  - ☐ Negative test
- Malformed calldata or nonzero value sent to the intended nonpayable wrapper is rejected before library entry.
  - ☐ Negative test
- A direct state-mutating `CALL` to deployed library bytecode is blocked by Solidity's runtime delegatecall guard.
  - ☐ Negative test
- Dispatch failure, `NotFreezer()`, corrupted-storage panic, or out-of-gas reverts the complete call frame.
  - ☐ Negative test

Function call analysis

- `this._getFreezeThawStorage()`
  - The generated trace misrenders the private internal helper as external; its Yul only assigns the fixed namespace to the storage pointer and makes no call.
- `EnumerableSet.remove($.freezers, account)`

    - This is an internal set operation whose membership result either permits the event or triggers `FreezeThawModule.NotFreezer()`, with no callback or reentrancy.
- **Manual call-path verification**: The path is `FrxUSDOFTUpgradeable.removeFreezer(account)` → nonpayable `onlyOwner` wrapper → `FreezeThawModule._removeFreezer(account)` → fixed-target delegatecall to `FreezeThawLib.removeFreezer(account)` → `_getFreezeThawStorage()` → `EnumerableSet.remove`; the generated FCA omitted the module-to-library delegatecall, and all failures propagate atomically.

Function: `thaw(address account)`

This function removes `account` from the fixed frozen set, ignores the removal result, and always emits `FreezeThawModule.AccountThawed(account)`. Only the owner-gated single and batch wrappers provide authorization; their linked-library delegatecalls execute in proxy context.

Inputs

- `account`
    - **Control**: The owner selects `account` directly or as an unchanged element of the caller-supplied batch.
    - **Constraints**: Any canonically encoded address is accepted, including a zero, duplicate, absent, or already thawed addresses.
    - **Impact**: It selects the frozen-set key at slot `0xf6192aebe04228448ed4fd2802d59c009f5324cdd6a110c65bd318373b6e1d00`, may trigger swap-and-pop removal, and is always emitted.

Branches and code coverage

Intended branches

- A present tested account is removed, and the frozen set becomes empty.
    - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Thaw_succeeds`)
- The thaw event is emitted.
    - ☐ Test coverage
- A present nonfinal element is replaced by the prior final element, indexes are updated, and enumeration order changes.
    - ☐ Test coverage
- An absent element leaves storage unchanged but still emits the thaw event.
    - ☐ Test coverage

- A nonempty batch removes each tested frozen account.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_ThawMany_succeeds`)
- The batch emits one thaw event per element; duplicate and empty-array behavior remains untested.
  - ☐ Test coverage
- Zero follows normal membership handling, and successful removal clears the freeze state for non-owner mint and burn paths.
  - ☐ Test coverage

Negative behavior

- A non-owner caller is rejected.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Thaw_OnlyOwner`)
- Reversion before delegation, mutation, or event emission is not instrumented.
  - ☐ Negative test
- Malformed direct or batch calldata reverts during ABI decoding, while any later batch failure atomically rolls back prior writes and logs.
  - ☐ Negative test
- The nonpayable wrappers reject nonzero value, although Solc 0.8.22 linked-library delegatecalls can preserve nonzero outer value from another caller.
  - ☐ Negative test
- A failed delegatecall, an out-of-gas condition, or an exception during removal causes all intermediate writes and logs to revert.
  - ☐ Negative test
- A direct library `CALL` is blocked by runtime protection, while an unrelated delegatecall can affect only the unrelated caller's storage.
  - ☐ Negative test

Function call analysis

- `this._getFreezeThawStorage()`
  - The trace renders this private pure helper as an external call. The helper returns the fixed namespaced storage slot without callbacks or caller control.
- `EnumerableSet.remove($.frozen, account)`
  - This internal operation uses current membership, indexes, length, and final element to remove by swap-and-pop, with its boolean result ignored.
- `FreezeThawLib.thaw(account)` via the resolved scoped wrapper paths

    - The `thaw` and `thawMany` wrappers enforce `onlyOwner` before making statically linked delegatecalls. Those calls mutate proxy storage, emit from proxy context, and propagate failures atomically.
- **Manual call-path verification**: The generated trace renders `_getFreezeThawStorage()` as an external call and omits both wrapper paths. Unrelated ASI builder failures concern Tempo `_quote` expressions in FraxOFTMintableAdapterUpgradeableTIP20.sol and FraxOFTUpgradeableTempo.sol.

Function: `thawMany(address[] accounts)`

This function removes ordered `accounts` from the active execution context's fixed frozen set and always emits `AccountThawed` per element. Removal is authorization-free and idempotent.

Inputs

- `accounts`
    - **Control**: Delegated calldata supplies each ordered set key and event argument, while `msg.sender` and preserved `msg.value` are unused.
    - **Constraints**: ABI decoding enforces valid array bounds and clean addresses, but not uniqueness, membership, zero-address exclusion, or zero preserved value.
    - **Impact**: Each element triggers a removal attempt and event, with array length controlling storage work, logs, and gas usage.

Branches and code coverage

Intended branches

- An empty array performs no removal and emits no event.
    - ☐ Test coverage
- A nonempty tested array processes and removes every frozen element.
    - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_ThawMany_succeeds`)
- The batch emits one event per iteration.
    - ☐ Test coverage
- Each tested frozen account is removed.
    - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_ThawMany_succeeds`)
- After each removal, `AccountThawed(account)` is emitted.
    - ☐ Test coverage
- An absent or previously removed duplicate account causes no membership change but

still emits the event.

- ☐ Test coverage

- The zero address (`address(0)`) is accepted as an ordinary key and always produces its event.
  - ☐ Test coverage

Negative behavior

- A direct `CALL` to the deployed library reverts under Solidity's state-mutating library protection.
  - ☐ Negative test
- Malformed array offsets, bounds, lengths, or dirty address encodings revert during ABI decoding.
  - ☐ Negative test
- Preserved nonzero value is accepted and unused here, while the intended nonpayable wrapper rejects it before delegation.
  - ☐ Negative test
- Nonempty static execution reverts on mutation or logging, while an empty array can return without either operation.
  - ☐ Negative test
- Gas exhaustion reverts atomically and rolls back all prior removals and logs.
  - ☐ Negative test
- Zero, absent, and duplicate addresses are accepted, so events do not prove prior membership.
  - ☐ Negative test
- The intended wrapper rejects a non-owner caller.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_ThawMany_OnlyOwner`)

Function call analysis

- `this.thaw(accounts[i]) -> this._getFreezeThawStorage()`
  - ASI misrenders both operations as external self-calls; they are internal, and the assembly helper only selects the fixed storage slot.
- `this.thaw(accounts[i]) -> EnumerableSet.remove($.frozen, account)`
  - The `EnumerableSet.remove` function is internal, its ignored boolean reflects membership, and it creates no callback or reentrancy edge.
- **Manual call-path verification**: The body makes no external calls. The intended fixed-link library `delegatecall`, omitted by ASI, preserves caller storage,

`address(this)`, `msg.sender`, and `msg.value`. It returns no application value and propagates failures atomically. ASI builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol:174` and `FraxOFTUpgradeableTempo.sol:48` do not affect this path.

5.12. Module: FrxUSD2.sol

Function: `addFreezer(address _freezer)`

Only the current owner can set `isFreezer[_freezer]` to `true` and emit `AddFreezer(_freezer)` without transferring ETH or tokens. A callable freezer may freeze `address(0)`, impairing the liveness of non-owner minting and burning, while thawing remains owner only.

Inputs

- `_freezer`
  - **Control**: The owner supplies this calldata address. The caller (`msg.sender`), current ownership, and existing mapping state also determine success.
  - **Constraints**: Any ABI-valid address, including zero or the owner, is accepted only when `isFreezer[_freezer]` is false, and nonzero `msg.value` is rejected.
  - **Impact**: The address becomes a mapping key and nonindexed event argument. The assignee gains `freeze` and `freezeMany` authority but not `thaw` or `thawMany`.

Branches and code coverage

Intended branches

- The owner sets the previously unset freezer mapping entry to `true`.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_addFreezer_successful`)
- `AddFreezer(_freezer)` is emitted.
  - ☐ Test coverage
- If `_freezer` is `address(0)` and its entry is unset, the function records and emits it even though the zero address cannot call the freeze functions.
  - ☐ Test coverage
- An assigned freezer can invoke `freeze` and `freezeMany` but has no role-based authority to thaw.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_freezer_can_freeze`, `FrxUSD_Mainnet_Compliance.test_freezer_can_freezeMany`,

FrxUSD_Mainnet_Compliance.test_freezer_cannot_thaw)

- The assigned freezer can freeze `address(0)`, which blocks non-owner mint and burn updates.

  ☐ Test coverage

Negative behavior

- A non-owner call reverts with the owner-authorization error.

  ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_addFreezer`)
- Ordering before mapping reads and effects is not independently asserted.

  ☐ Negative test
- An owner call for an existing freezer reverts with `AlreadyFreezer()`.

  ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_addFreezer_reverts_if_already_freezer`)
- Absence of a write or event is not independently asserted.

  ☐ Negative test
- A call carrying nonzero `msg.value` is rejected by the nonpayable entry check before authorization or effects.

  ☐ Negative test

Function call analysis

- **Manual call-path verification**: Concrete FrxUSD reaches only the internal owner check, mapping read, conditional revert, mapping write, and event emission; `_freezer` is never called, checks precede effects, failures roll back state and logs, `tx.origin` is unused, and no callback or reentrancy edge exists.

Function: `addMinter(address minter_address)`

This function authorizes a nonzero, currently unauthorized address in FrxUSD2's `minters` mapping, appends it to `minters_array`, and emits `MinterAdded`. Authorization requires exactly `msg.sender == timelock`. Code presence and delay enforcement remain deployment and governance assumptions.

Inputs

- `minter_address`
  - **Control**: The caller supplies this unchanged mapping key, appended value, and event argument.

- **Constraints**: It must be nonzero and currently unauthorized, while code presence, interfaces, and historical uniqueness are unchecked.
- **Impact**: Success authorizes the address, grows `minters_array`, emits `MinterAdded(minter_address)`, and increases the size of the array scanned during removal.

- Implicit state includes `msg.sender`, `timelock`, `minters`, and `minters_array`. A zero timelock admits no ordinary EVM caller, and the nonpayable dispatcher rejects nonzero `msg.value`.

Branches and code coverage

Intended branches

- An authorized timelock call with a nonzero, unauthorized candidate sets the minter mapping entry to `true`.

  ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_addMinter_successful`)

- The successful addition appends the candidate to `minters_array` and emits `MinterAdded`.

  ☐ Test coverage

- Re-addition after removal: a removed address can be re-added beyond its existing tombstone, and repeated privileged cycles continually grow the linearly scanned array until removal may become impractical or exceed block gas limits.

  ☐ Test coverage

Negative behavior

- A caller other than the configured timelock reverts with `"Only timelock can add minters"`; an owner cannot add before a timelock is set.

  ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_timelock_can_addMinter`, `FrxUSD_Mainnet_Compliance.test_addMinter_reverts_before_timelock_set`)

- Zero candidate: an authorized call with `minter_address == address(0)` reverts with `"Zero address detected"` before any write or event.

  ☐ Negative test

- Already authorized candidate: an authorized, nonzero candidate whose mapping value is `true` reverts with `"Address already exists"` before any write or event.

  ☐ Negative test

- Unexpected ETH: a call with nonzero `msg.value` reverts in Solidity's nonpayable dispatcher before entering the function body.

  ☐ Negative test

- Exceptional failure: any exceptional failure during the mapping write, array append, or event emission atomically rolls back all writes and logs.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The inherited path FrxUSD → FrxUSD3 → `FrxUSD2.addMinter` executes this exact body without overrides, helpers, callbacks, token or ETH movement, or any reentrancy point.

Function: `burn(address _owner, uint256 _amount)`

Only the current Ownable owner may burn frxUSD from any account, and a zero amount selects that account's full balance. Pause and freeze checks do not restrict this owner-only path. Proxy execution reads preserved proxy ownership, while the directly deployed implementation initializes the owner to `address(1)`.

Inputs

- `_owner`
  - **Control**: The current owner selects any account whose balance will be burned.
  - **Constraints**: The address must be nonzero, and the selected amount cannot exceed its balance.
  - **Impact**: Success reduces the account's balance and total supply and emits it as the `Transfer` sender.
- `_amount`
  - **Control**: The current owner supplies an explicit amount, while zero selects `_owner`'s current balance.
  - **Constraints**: A nonzero selected amount cannot exceed `_owner`'s balance.
  - **Impact**: It determines the balance and supply reduction, including a successful zero-value burn for an empty account.

Branches and code coverage

Intended branches

- Zero `msg.value` passes nonpayable dispatch and reaches authorization.
  - ☐ Test coverage
- The current accepted owner satisfies `onlyOwner` in the active execution context.

- ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burn_tokens`)

- Zero `_amount` reads and burns `_owner`'s full current balance.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burn_tokens`)
- Nonzero `_amount` is forwarded unchanged to `_burn`.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burn_tokens`)
- Authorized execution bypasses `FrxUSD2._update` pause and freeze checks.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burn_tokens_when_frozen`, `FrxUSD_Mainnet_Compliance.test_can_burn_tokens_when_paused`)
- A valid nonzero burn reduces `_owner`'s balance.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burn_tokens`)
- A valid burn reduces the total supply and emits `Transfer`.
  - ☐ Test coverage
- Burning the full balance of an empty nonzero account succeeds and emits a zero-value `Transfer`.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` reverts before authorization or body execution.
  - ☐ Negative test
- A caller other than the current owner reverts.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_burn`)
- Ordering before the function body is not independently asserted.
  - ☐ Negative test
- Renounced ownership makes the function inaccessible to ordinary external callers.
  - ☐ Negative test
- A zero `_owner` reverts with `ERC20InvalidSender(address(0))`.
  - ☐ Negative test
- An amount exceeding `_owner`'s balance reverts with `ERC20InsufficientBalance`.
  - ☐ Negative test
- Pause or freeze state does not cause failure after successful owner authorization.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_can_burn_tokens_when_frozen`, `FrxUSD_Mainnet_Compliance.test_can_burn_tokens_when_paused`)

Function call analysis

- **Manual call-path verification**: The internal route is `burn` → `onlyOwner` → optional `balanceOf` → `_burn` → `FrxUSD2._update` → `ERC20._update`; no sibling sfrxUSD path, ETH transfer, callback, runtime delegatecall, or reentrancy edge is reachable.

Function: `burnMany(address[] _owners, uint256[] _amounts)`

The `burnMany` function lets the current OpenZeppelin Ownable owner confiscate holder balances in batch without the holders' authorization or allowances. Zero amounts resolve to each address's then-current balance, and the function returns nothing.

Inputs

- `_owners`
  - **Control**: The effective owner controls every address, while `onlyOwner` rejects all other callers and `tx.origin` is unused.
  - **Constraints**: The array may be empty or contain duplicates, but its length must match `_amounts` and every processed address must be nonzero.
  - **Impact**: Each address selects the balance debited by `_burn`, with any failure reverting all prior writes and logs.
- `_amounts`
  - **Control**: The effective owner controls every amount, while `msg.value` must be zero and no allowance, signature, or external value contributes.
  - **Constraints**: Zero resolves to the owner's current balance, nonzero passes unchanged, and the resolved amount cannot exceed the then-current balance.
  - **Impact**: The resolved amount reduces the selected balance and total supply and emits `Transfer` to `address(0)` without consuming allowance or emitting `Burn`.

Branches and code coverage

Intended branches

- The current owner passes the inherited `onlyOwner`, using implementation storage directly or preserved proxy storage under delegation.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens`)
- Equal-length arrays pass the pre-loop `ArrayMisMatch()` check.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens`)
- Empty equal-length arrays return without state changes or events.

☐ Test coverage

- A zero amount resolves to and burns the address's then-current full balance.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens`)

- A nonzero amount is burned unchanged when the source is nonzero and sufficiently funded.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_burnMany_with_specific_amounts`)

- A resolved zero amount from a nonzero source leaves balances and supply unchanged but emits a zero-value `Transfer`.

☐ Test coverage

- Duplicate addresses execute sequentially, so later entries observe balances remaining after earlier burns.

☐ Test coverage

- Owner-authorized batch burns succeed for tested frozen source accounts and while paused.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens_when_frozen`, `FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens_when_paused`)

- Frozen owner/caller and frozen `address(0)` combinations remain untested.

☐ Test coverage

- Inherited ERC-20 accounting reduces each tested source balance without allowance consumption.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens`, `FrxUSD_Mainnet_Compliance.test_burnMany_with_specific_amounts`)

Negative behavior

- A non-owner caller is rejected.

☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_burnMany`)

- Ordering before length checks and burns is not independently asserted.

☐ Negative test

- Unequal array lengths revert with `ArrayMisMatch()`.

☑ Negative test (`FrxUSD_Mainnet_Compliance.test_can_burnMany_tokens_reverts_array_misma`

- Ordering before iteration is not instrumented.

☐ Negative test

- Any zero source address reverts in inherited `_burn`, including a zero-value burn.

- ☐ Negative test
- Any explicit or cumulative burn exceeding the then-current balance reverts in ERC-20 accounting.
  - ☐ Negative test
- Malformed ABI input or out-of-gas during decoding, memory expansion, or iteration reverts atomically.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by Solidity's nonpayable dispatcher before authorization or body execution.
  - ☐ Negative test
- A revert at any index discards all earlier balance changes, supply changes, memory substitutions, and `Transfer` logs.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `onlyOwner` modifier, `balanceOf`, `_burn`, `FrxUSD2._update`, and OpenZeppelin accounting remain internal with no callbacks or reentrancy. Proxy correctness and owner-key availability, security, and monitoring remain critical.

Function: `freeze(address _owner)`

Authorized freezers or the current owner set `isFrozen[_owner] = true` and emit `AccountFrozen(_owner)`. Repeated freezes succeed and reemit the event. No ETH or tokens move.

Inputs

- `_owner`
  - **Control**: The caller selects this calldata address.
  - **Constraints**: Any canonical address is accepted, including zero and already frozen addresses, with no paused-state check.
  - **Impact**: It is used unchanged as the mapping key and event argument, with `address(0)` blocking non-owner–operated mint and burn transitions while preserving the owner's explicit `_update` bypass.
- `msg.sender`
  - **Control**: The immediate calling account or contract determines this value.
  - **Constraints**: It must be an enabled freezer or the current owner, with freezer

status short-circuiting `owner()`.
    - **Impact**: It controls authorization but is neither stored nor emitted.
- `msg.value`
    - **Control**: The immediate caller supplies this value.
    - **Constraints**: The nonpayable entry point requires zero value.
    - **Impact**: Nonzero value reverts before any storage write or event.
- `tx.origin`
    - **Control**: The originating transaction account determines this value.
    - **Constraints**: No constraint applies because this path never reads it.
    - **Impact**: It has no effect.

Branches and code coverage

Intended branches

- A configured non-owner freezer is authorized and can freeze an account.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_freezer_can_freeze`)
- If `isFreezer[msg.sender]` is false and `msg.sender == owner()`, authorization passes and the call proceeds to `_freeze(_owner)`.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_upgrade_and_freeze_successful`)
- An unfrozen account changes to frozen.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_upgrade_and_freeze_successful`, `FrxUSD_Mainnet_Compliance.test_freezer_can_freeze`)
- `AccountFrozen(_owner)` is emitted.
    - ☐ Test coverage
- If `isFrozen[_owner]` is already true, it is rewritten to true and `AccountFrozen(_owner)` is emitted again.
    - ☐ Test coverage
- Freezing `address(0)` succeeds, blocking non-owner–operated mint and burn transitions while preserving the owner's explicit `_update` bypass.
    - ☐ Test coverage

Negative behavior

- A caller that is neither freezer nor owner reverts with `NotFreezer()`.
    - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_freeze`,

FrxUSD_Mainnet_Compliance.test_non_freezer_cannot_freeze)

- The test does not separately assert mutation ordering.
  - ☐ Negative test
- Nonzero `msg.value` is rejected before authorization or mutation.
  - ☐ Negative test
- Short or noncanonical calldata fails ABI decoding before authorization or mutation.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `FrxUSD.freeze(address)` call resolves through inherited `FrxUSD2.freeze(address)` to the conditional internal `owner()` and fixed nonvirtual `FrxUSD2._freeze(address)`, which writes storage and logs without callbacks or reentrancy. Any later EVM revert atomically rolls both back.

Function: `freezeMany(address[] _owners)`

This function freezes every supplied address after freezer-or-owner authorization by setting `isFrozen` and emitting `AccountFrozen`. Empty, duplicate, already frozen, and zero addresses are accepted; the function returns nothing and moves no value or tokens.

Inputs

- `_owners`
  - **Control**: The caller controls the array's length, order, duplicates, existing entries, and address values.
  - **Constraints**: The array may be empty and has no uniqueness, nonzero-address, already-frozen, or explicit length requirement.
  - **Impact**: Its length determines loop gas, while each element becomes a mapping key and event value.

Branches and code coverage

Intended branches

- A configured non-owner freezer is authorized to freeze a batch.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_freezer_can_freezeMany`)
- A nonfreezer is authorized when `msg.sender == owner()`.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_upgrade_and_freezeMany`)

- An authorized empty array completes without storage writes or events.

☐ Test coverage

- Each tested array element is marked frozen.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_upgrade_and_freezeMany`, `FrxUSD_Mainnet_Compliance.test_freezer_can_freezeMany`)

- `AccountFrozen` is emitted for each element.

☐ Test coverage

- Duplicate and already frozen entries repeat the write and event without rejection.

☐ Test coverage

- An `address(0)` entry is accepted, blocking later non-owner mint and burn `_update` paths, while the owner bypasses those checks and alone can thaw it.

☐ Test coverage

Negative behavior

- An unauthorized caller reverts with `NotFreezer()`.

☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_freezeMany`, `FrxUSD_Mainnet_Compliance.test_non_freezer_cannot_freezeMany`)

- Absence of every write and event is not independently asserted.

☐ Negative test

- Malformed calldata or nonzero `msg.value` is rejected before the function body without state or log effects.

☐ Negative test

- An oversized authorized batch may run out of gas, atomically rolling back all writes and logs.

☐ Negative test

Function call analysis

- **Manual call-path verification**: The `owner()` function only reads inherited storage, while the nonvirtual `_freeze` function writes the mapping and then emits a log. Neither path transfers value, invokes callbacks, or permits reentrancy.

Function: `minter_burn_from(address b_address, uint256 b_amount)`

Authorized minters burn frxUSD from `b_address` through `ERC20Burnable.burnFrom`. Non-owner minters remain subject to pause and freeze controls, while authorization, allowance, nonzero-source, and balance requirements always apply.

Inputs

- `b_address`
  - **Control**: The caller selects the source address independently of minter authorization.
  - **Constraints**: Success requires a nonzero, sufficiently funded source with adequate allowance and, for non-owners, no applicable freeze.
  - **Impact**: It identifies the allowance owner, burned balance, compliance subject, and event source.
- `b_amount`
  - **Control**: The authorized caller selects the unmodified burn amount.
  - **Constraints**: The amount may be zero but cannot exceed the source balance or finite allowance.
  - **Impact**: It reduces finite allowance, source balance, and total supply and appears unchanged in both events.
- `msg.sender`
  - **Control**: The immediate caller supplies this identity without trusted-forwarder substitution.
  - **Constraints**: The caller must be an authorized minter and allowance spender, with non-owners also subject to pause and freeze checks.
  - **Impact**: It selects authorization, allowance, owner bypass, caller freeze, and the `TokenMinterBurned` minter field.
- `msg.value`
  - **Control**: The calling transaction or contract selects the attached native value.
  - **Constraints**: It must be zero because the function is nonpayable.
  - **Impact**: It is neither read nor transferred.
- Relevant contract state
  - **Control**: Existing authorization, allowance, ownership, compliance, balance, and supply state comes from separate token or governance paths.
  - **Constraints**: Authorization precedes allowance processing, compliance checks, and inherited balance accounting.
  - **Impact**: State determines success, allowance handling, owner bypass, and accounting writes, all of which revert atomically on failure.

Branches and code coverage

Intended branches

- With authorization, sufficient allowance and balance, a nonzero source, and satisfied compliance checks, the burn emits `Transfer` before `TokenMinterBurned`.
  - ☐ Test coverage
- An allowance of `type(uint256).max` remains unchanged while the burn proceeds.
  - ☐ Test coverage
- A sufficient finite allowance decreases by `b_amount` without emitting `Approval`.
  - ☐ Test coverage
- An authorized owner-minter bypasses pause and freeze checks but not authorization, allowance, source, or balance requirements.
  - ☐ Test coverage
- An authorized non-owner minter burns an approved source balance while unpaused and unfrozen.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_minter_can_burn_from`)
- A zero-amount burn may succeed and emit zero-value events but still executes every authorization and compliance check.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` is rejected before the function body executes.
  - ☐ Negative test
- An unauthorized caller reverts with `"Only minters"`.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_non_minter_cannot_burn_from`)
- Ordering before accounting effects is not independently asserted.
  - ☐ Negative test
- An insufficient finite allowance reverts with `ERC20InsufficientAllowance` and rolls back all effects.
  - ☐ Negative test
- A zero source reverts with `ERC20InsufficientAllowance`, `ERC20InvalidApprover`, or, for infinite allowance, `ERC20InvalidSender`.
  - ☐ Negative test
- A paused call by a non-owner minter reverts with `IsPaused()` and rolls back any allowance write.

- ☐ Negative test
- A frozen source, zero address, or caller reverts for non-owner minters with `IsFrozen()`; a frozen zero address blocks every such burn.
  - ☐ Negative test
- An insufficient source balance reverts with `ERC20InsufficientBalance` and atomically rolls back prior effects.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `burnFrom` function reaches `_spendAllowance`, `_burn`, `FrxUSD2._update`, and `ERC20._update` without external control transfer. Append-only linear administration may impair mapping-based minter revocation. The configured timelock contract and delay were not verified.

Function: `minter_mint(address m_address, uint256 m_amount)`

Authorized minters mint any `uint256` amount, including zero, to nonzero recipients. Successful calls increase supply and balance and emit `Transfer` and then `TokenMinterMinted`. Owner-minters bypass pause and freeze checks, and the function returns no value.

Inputs

- `m_address`
  - **Control**: The authorized caller selects the recipient.
  - **Constraints**: It must be nonzero and, for non-owner minters, unfrozen.
  - **Impact**: Its balance increases by `m_amount`, and it appears in both events.
- `m_amount`
  - **Control**: The authorized caller selects any `uint256` amount.
  - **Constraints**: Zero is allowed, but checked supply addition must not overflow.
  - **Impact**: It increases supply and recipient balance and appears in both events.
- `msg.sender`
  - **Control**: The immediate EOA or contract caller determines `msg.sender`.
  - **Constraints**: It must be an authorized minter; only owner-minters bypass compliance checks.
  - **Impact**: It keys authorization and freezing, is compared with `owner()`, and is emitted in `TokenMinterMinted`.
- `minters[msg.sender]`

    - **Control**: The timelock adds minters, while the owner removes them.
    - **Constraints**: It must be true before the function body executes.
    - **Impact**: It gates minting and is not modified here.
- `owner()`
    - **Control**: Inherited ownership state determines the current owner.
    - **Constraints**: Ownership does not authorize minting but enables compliance bypass for an authorized minter.
    - **Impact**: It is internally compared with `msg.sender`.
- `isPaused`
    - **Control**: The owner controls pause state.
    - **Constraints**: A true value reverts non-owner minting with `IsPaused()`.
    - **Impact**: It gates non-owner minters without being modified.
- `isFrozen[m_address]`, `isFrozen[address(0)]`, and `isFrozen[msg.sender]`
    - **Control**: The owner or a freezer sets freezes, while only the owner thaws them.
    - **Constraints**: Non-owner paths check recipient, zero address, then caller and revert with `IsFrozen()`.
    - **Impact**: These entries gate minting, with `isFrozen[address(0)]` globally blocking non-owner minters.
- Current total supply and `balanceOf(m_address)`
    - **Control**: Prior ERC-20 state determines both values.
    - **Constraints**: Checked supply addition reverts on overflow.
    - **Impact**: Both increase by `m_amount` on success and remain unchanged on revert.
- `msg.value`
    - **Control**: The external caller selects attached native value.
    - **Constraints**: Nonzero value is rejected because the function is nonpayable.
    - **Impact**: No successful path reads, stores, or forwards it.
- `tx.origin`
    - **Control**: The transaction-originating EOA determines it.
    - **Constraints**: No constraint uses it.
    - **Impact**: It has no effect because authorization uses `msg.sender`.

Branches and code coverage

Intended branches

- Authorized caller: an authorized `msg.sender` passes `onlyMinters` and reaches `_mint`.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_minter_can_mint`)

- Nonzero recipient: a nonzero recipient passes `ERC20._mint` validation and reaches `_update`.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_minter_can_mint`)

- Owner-minter bypass: an authorized owner-minter bypasses all pause and freeze checks.

☐ Test coverage

- An unpaused, unfrozen non-owner minter reaches the ERC-20 update.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_minter_can_mint`)

- A positive representable amount increases total supply and the recipient balance.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_minter_can_mint`)

- The successful mint emits the ERC-20 and minter events in order.

☐ Test coverage

- Zero amount: zero still runs all checks and emits both events without numerical state changes.

☐ Test coverage

Negative behavior

- A non-minter reverts with `"Only minters"`.

☑ Negative test (`FrxUSD_Mainnet_Compliance.test_non_minter_cannot_mint`)

- Ordering before `_mint` is not instrumented.

☐ Negative test

- Zero recipient: the zero recipient reverts with `ERC20InvalidReceiver(address(0))` before `_update`.

☐ Negative test

- Paused non-owner minter: a paused non-owner path reverts with `IsPaused()` before freeze checks.

☐ Negative test

- Frozen recipient: a frozen recipient causes the non-owner path to revert with `IsFrozen()`.

☐ Negative test

- Frozen zero address: a frozen zero address reverts non-owner minting with `IsFrozen()`.

☐ Negative test

- Frozen caller: a frozen non-owner minter reverts with `IsFrozen()` after earlier checks pass.

☐ Negative test

- Supply overflow: overflowing `totalSupply + m_amount` reverts with an arithmetic panic and rolls back atomically.

☐ Negative test

- Nonzero native value: nonpayable dispatch rejects nonzero `msg.value` before function execution.

☐ Negative test

Function call analysis

- **Manual call-path verification**: The `onlyMinters` modifier leads through `ERC20._mint`, `FrxUSD2._update`, internal `owner()` access, and `ERC20._update` without external calls, receiver hooks, native-value movement, or reentrant control transfer.

Findings

- **Frozen zero address blocks non-owner minting**: Freezing `address(0)` blocks minting by every authorized non-owner minter, while authorized owner-minters bypass the freeze.
- **Costly minter revocation**: Historical `minters_array` growth and tombstones can make owner revocation too costly to execute, extending a minter's access to this function.

Function: `pause()`

The owner-only `pause()` sets `isPaused = true`, emits `Paused()`, returns no data, and permits repeated calls. Direct execution uses implementation storage, while intended legacy-proxy dispatch preserves sender/value, uses proxy ownership/storage, and emits from the proxy.

Branches and code coverage

Intended branches

- An authorized owner sets `isPaused` to `true`.

☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_upgrade_and_pause_successful`)

- A successful call emits `Paused()` and returns no data.

☐ Test coverage

- While paused, a non-owner transfer update reverts with `IsPaused()`, whereas an owner burn succeeds.

☑ Test coverage

(FrxUSD_Mainnet_Compliance.test_transfer_when_paused_reverts, FrxUSD_Mainnet_Compliance.test_can_burn_tokens_when_paused)

- Through the tested legacy proxy, pausing uses preserved proxy ownership and storage and changes only the expected pause slot.

  ☑ Test coverage (FrxUSD_Mainnet_Compliance.test_storage_layout_change_when_paused)

- Because pause() is nonpayable, nonzero msg.value reverts before execution in both direct and delegated contexts without changing state or logs.

  ☐ Test coverage

Negative behavior

- A caller other than the owner reverts with the owner-authorization error.

  ☑ Negative test (FrxUSD_Mainnet_Compliance.test_only_owner_can_pause)

- Ordering before assignment or event emission is not independently asserted.

  ☐ Negative test

- An authorized call while already paused succeeds, rewrites true, and emits another Paused() event.

  ☐ Negative test

- The function performs no transfer, mint, burn, approval, confiscation, or supply update. Minter growth, zero-address freezing, and timelock semantics are unrelated.

  ☐ Negative test

- Any authorization failure, nonpayable rejection, or other revert atomically removes all writes and logs from the call frame.

  ☐ Negative test

Function call analysis

- **Manual call-path verification**: The FrxUSD → FrxUSD3 → FrxUSD2.pause path performs only internal ownership checks, a storage write, and a local event. Proxy delegatecall is inbound, and pause() creates no outbound call, callback, token/value movement, or reentrancy point.

Function: removeFreezer(address _freezer)

The owner-only removeFreezer revokes one mapping-based delegated freezer and emits RemoveFreezer. The scoped FrxUSD3 and FrxUSD contracts inherit it unchanged, while sfrxUSD does not expose it. It changes no balances, allowances, supply, frozen entries, timelocks, nonces, or native balances.

Inputs

- `_freezer`
  - **Control**: The owner supplies this mapping key and nonindexed event argument without using it as a call target or transfer endpoint.
  - **Constraints**: The caller (`msg.sender`) must equal `owner()`, and `isFreezer[_freezer]` must be true. The zero address is allowed if previously added.
  - **Impact**: Success sets only `isFreezer[_freezer]` to false in the active implementation or proxy storage and revokes delegated freezing authority.

The function is nonpayable, proxy delegatecall preserves `msg.sender`, and `tx.origin` is unused.

Branches and code coverage

Intended branches

- The owner changes a current freezer mapping entry from true to false.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_removeFreezer_successful`)
- A successful removal emits `RemoveFreezer(_freezer)` and returns no value.
  - ☐ Test coverage
- A removed freezer can no longer call `freeze` unless re-added or independently authorized as owner.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_removed_freezer_cannot_freeze`)
- Removing a freezer leaves existing account freezes unchanged, including `isFrozen[address(0)]`.
  - ☐ Test coverage

Negative behavior

- A non-owner reverts with the owner-authorization error.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_removeFreezer`)
- Absence of writes and events is not independently asserted.
  - ☐ Negative test
- An owner call for the tested nonmember reverts with `NotFreezer()`.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_removeFreezer_reverts_if_not_freezer`)
- An unset zero-address entry is not separately exercised.

- ☐ Negative test

- Nonzero native value: nonpayable dispatch rejects `msg.value != 0` before authorization or body execution.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `FrxUSD.removeFreezer` call resolves to inherited `FrxUSD2.removeFreezer`, performs local owner authorization, reads and conditionally clears `isFreezer[_freezer]`, and emits the event without callbacks, transfers, return-data dependencies, or outgoing calls. Inbound proxy delegatecall only selects proxy storage.

Function: `removeMinter(address minter_address)`

Only the current owner can revoke a nonzero active minter, disabling its `minter_mint` and `minter_burn_from` access. The mapping is authoritative; removal tombstones the first array match and emits `MinterRemoved`.

Inputs

- `minter_address`
  - **Control**: Supplied directly by the caller in calldata.
  - **Constraints**: Must be nonzero and mapped true, and no array match is required.
  - **Impact**: Selects the mapping deletion, first matching array tombstone, and event data.
- `msg.sender` (implicit)
  - **Control**: Determined by the immediate caller.
  - **Constraints**: Inherited `onlyOwner` requires it to equal the current owner.
  - **Impact**: Controls whether validation, storage writes, scanning, and emission can occur.
- `msg.value` (implicit)
  - **Control**: Chosen by the caller.
  - **Constraints**: Must be zero because the function is nonpayable.
  - **Impact**: Any nonzero value reverts before persistent effects.
- Current owner storage (implicit)
  - **Control**: Established through the inherited ownership life cycle.
  - **Constraints**: Must equal `msg.sender`.

    - **Impact**: Defines the sole authorized caller and is not modified here.
- `minters[minter_address]` (implicit storage input)
    - **Control**: Set by minter-management transitions or inherited deployment state.
    - **Constraints**: Must be true after the zero-address check.
    - **Impact**: Its deletion immediately revokes the selected minter's authorization.
- `minters_array` contents and length (implicit storage inputs)
    - **Control**: Accumulate through additions, removals, and any inherited state.
    - **Constraints**: The loop scans from zero and stops at the first matching entry.
    - **Impact**: Determines the tombstoned slot and linear gas cost.
- Available execution gas (implicit)
    - **Control**: Bounded by transaction gas and the block gas limit.
    - **Constraints**: Must cover authorization, validation, scanning, writes, and emission.
    - **Impact**: Insufficient gas atomically restores storage and discards logs.

Branches and code coverage

Intended branches

- Owner authorization succeeds: the `onlyOwner` modifier accepts the current owner and enters the body.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_removeMinter_successful`)
- Valid nonzero active minter: both checks pass, and the mapping entry is deleted.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_removeMinter_successful`)
- Nonmatching visited slot: the loop skips the slot and advances.
    - ☐ Test coverage
- First array match: the slot becomes `address(0)`, and `break` stops the scan.
    - ☐ Test coverage
- Scan completes without a match: in inconsistent inherited or external state, deletion and emission still succeed.
    - ☐ Test coverage
- Successful completion: the `MinterRemoved(minter_address)` event is emitted, and no data is returned.
    - ☐ Test coverage

Negative behavior

- A non-owner caller reverts.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_removeMinter`)
- Ordering before validation, writes, and emission is not independently asserted.
  - ☐ Negative test
- Zero address: the call reverts with `"Zero address detected"`.
  - ☐ Negative test
- Never added or already removed address: the call reverts with `"Address nonexistant"`.
  - ☐ Negative test
- Insufficient gas during the linear scan: permanent append/tombstone growth can make late removal exceed the block gas limit and revert atomically.
  - ☐ Negative test
- Nonzero ETH: nonpayable dispatch rejects the call without persistent effects.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: Source review confirms inherited `onlyOwner` and local storage logic only, with no hooks, callbacks, token or ETH movement, or reentrancy edge.

Function: `setTimelock(address _timelock)`

This function sets or rotates authority for `addMinter` and later rotations. Supported legacy proxies use rotation, while the separate implementation has zero timelock and owner `address(1)` without initializing proxy storage.

Inputs

- `_timelock`
  - **Control**: The caller selects the nonzero address, with the owner authorized only while `timelock` is zero and the current timelock authorized thereafter.
  - **Constraints**: Authorization requires `msg.sender == timelock || (timelock == address(0) && msg.sender == owner())`, `_timelock` must be nonzero, and `msg.value` must be zero.

- **Impact**: The function emits `TimelockChanged(timelock, _timelock)` and stores `_timelock` as the authority for `addMinter` and future rotations.

Branches and code coverage

Intended branches

- Source-level bootstrap success, deployment-conditional: when `timelock` is zero, the owner may set a nonzero timelock, though supported legacy proxies already use initialized timelocks.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_setTimelock_owner_bootstrap`)
- Rotation success: the current nonzero timelock may replace itself with another nonzero address.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_setTimelock_rotated_by_timelock`)
- Self-assignment success: the current timelock may assign itself, emitting identical old and new addresses without changing effective authority.
  - ☐ Test coverage

Negative behavior

- After bootstrap, the former owner cannot rotate a nonzero timelock and receives the expected authorization revert.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_owner_cannot_change_timelock_after_set`)
- The test does not separately assert unchanged storage or logs.
  - ☐ Negative test
- Zero-address proposal: an authorized caller proposing `address(0)` reverts with `"Zero address detected"` before the event or write.
  - ☐ Negative test
- Nonpayable boundary: a call carrying nonzero `msg.value` reverts at dispatch before executing the function body.
  - ☐ Negative test
- Externally owned or non-timelocked destination: any nonzero EOA or contract without enforced delay is accepted as the new timelock.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `owner()` call is an internal inherited read, event emission is a log, assignment does not invoke `_timelock`, and any failure rolls back the complete call.

Function: `thaw(address _owner)`

Current-owner authorization gates `_thaw(_owner)`, which clears `isFrozen[_owner]` and emits `AccountThawed(_owner)`. Any address, including zero or an already unfrozen account, is accepted.

Inputs

- `_owner`
  - **Control**: The current owner selects any ABI-valid address independently of `msg.sender`.
  - **Constraints**: No zero-address, frozen-state, uniqueness, or caller-relationship check applies, while nonzero `msg.value` is rejected and `tx.origin` is unused.
  - **Impact**: The value selects the cleared mapping entry and the unindexed event argument. Thawing the zero address removes the freeze condition that blocks non-owner mint and burn updates, subject to other checks.

Branches and code coverage

Intended branches

- Authorization success calls `_thaw(_owner)`, writes `isFrozen[_owner] = false`, emits `AccountThawed(_owner)`, and returns empty data.
  - ☐ Test coverage
- A frozen entry changes from true to false, after which transfers from that account succeed.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_thaw_restores_transfer`)
- Thawing emits `AccountThawed(_owner)`.
  - ☐ Test coverage
- An already unfrozen entry is rewritten to `false` and still emits because no `NotFrozen` check exists.
  - ☐ Test coverage
- The zero address is accepted and cleared, removing the freeze condition blocking non-owner mint and burn updates while other checks remain.

- ☐ Test coverage

Negative behavior

- A non-owner, including an assigned freezer, cannot thaw.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_thaw`, `FrxUSD_Mainnet_Compliance.test_freezer_cannot_thaw`)
- Absence of a write or event is not independently asserted.
  - ☐ Negative test
- Nonzero `msg.value` causes a nonpayable rejection before modifier or body execution.
  - ☐ Negative test
- A zero or already unfrozen address does not cause a revert.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `thaw` function performs inherited owner authorization and invokes the nonvirtual internal `_thaw` function, which writes once and emits without callbacks or reentrancy. Proxy delegatecall routing may occur before entry, outside this body, while preserving caller context and using proxy storage.

Function: `thawMany(address[] _owners)`

Owner-only batch unfreezing writes `false` and emits `AccountThawed` for every supplied occurrence. Proxy execution uses proxy owner/storage, while the live implementation is intentionally locked with owner `address(1)`.

Inputs

- `_owners`
  - **Control**: The caller selects the array length, order, duplicates, and address elements.
  - **Constraints**: Empty arrays, zero addresses, duplicates, and already thawed addresses are accepted, subject to valid ABI encoding and available gas.
  - **Impact**: Each occurrence writes `isFrozen[address] = false` and emits `AccountThawed`, while an empty array has no effects.
- `msg.sender`
  - **Control**: The immediate caller sets `msg.sender`, which delegate execution preserves through a proxy.
  - **Constraints**: It must equal the stored `owner()`, so freezer authority alone is

insufficient.
    - **Impact**: It determines whether execution reaches the loop but does not otherwise affect `_thaw`.
- `msg.value`
    - **Control**: The caller selects the value attached to the call.
    - **Constraints**: It must be zero because the function is nonpayable.
    - **Impact**: Nonzero value rejects the call before any write or event.
- `tx.origin`
    - **Control**: The transaction originator determines `tx.origin`.
    - **Constraints**: This path imposes no constraint on it.
    - **Impact**: It is unused and has no effect on authorization or state.

Branches and code coverage

Intended branches

- Authorization success: when `msg.sender == owner()`, execution reaches the batch loop.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_thawMany_restores_transfers`)
- Empty batch: zero elements produce no writes or logs and return.
    - ☐ Test coverage
- Each tested batch element is marked unfrozen, and both accounts can transfer afterward.
    - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_thawMany_restores_transfers`)
- Each thawed element is logged before the function returns.
    - ☐ Test coverage
- Unvalidated element cases: zero, duplicate, and already thawed addresses are processed per occurrence. Thawing `address(0)` removes the zero-address freeze blocking non-owner mint/burn updates, though pause or other freezes may still revert.
    - ☐ Test coverage

Negative behavior

- A non-owner, including an assigned freezer, cannot thaw a batch.
    - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_thawMany`, `FrxUSD_Mainnet_Compliance.test_freezer_cannot_thawMany`)
- Ordering before every loop effect is not independently asserted.

- ☐ Negative test

- Nonpayable rejection: nonzero `msg.value` reverts without persisted writes or logs.
  - ☐ Negative test
- Invalid or infeasible batches: malformed calldata, memory failure, insufficient gas, or later reversion atomically rolls back all earlier effects.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The resolved path is FrxUSD → FrxUSD3 → `FrxUSD2.thawMany` → internal `_thaw`. Authorization, storage writes, and events remain internal, with no callback or reentrancy edge, and proxy delegate execution preserves the caller while applying writes to proxy storage.

Function: `unpause()`

The `unpause()` function lets the stored owner set `isPaused` to `false` and emit `Unpaused()`, even if already unpaused. It performs no token operation and returns empty data.

The function takes no explicit arguments. The caller (`msg.sender`) is checked against owner storage, `tx.origin` is unused, and the prior `isPaused` value does not affect execution.

The call value (`msg.value`) must be zero. Direct construction sets the owner to `address(1)`, while proxy execution uses preserved ownership storage and administrator-selected implementation code.

Branches and code coverage

Intended branches

- An authorized owner changes `isPaused` from true to false, after which transfers succeed.
  - ☑ Test coverage (`FrxUSD_Mainnet_Compliance.test_unpause_restores_transfer`)
- The successful call emits `Unpaused()` and returns empty data.
  - ☐ Test coverage
- Authorized owner while already unpaused: an authorized zero-value call writes `false` again, emits `Unpaused()`, and returns empty data.
  - ☐ Test coverage
- Zero call value: with `msg.value == 0`, authorization proceeds independently of `tx.origin` or calldata arguments.

- ☐ Test coverage

Negative behavior

- A caller other than the stored owner reverts with the owner-authorization error.
  - ☑ Negative test (`FrxUSD_Mainnet_Compliance.test_only_owner_can_unpause`)
- Unchanged state and absence of an event are not independently asserted.
  - ☐ Negative test
- Nonzero native value: a nonzero-value call reverts before authorization, retaining no ETH and leaving state and logs unchanged.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The path consists only of the nonpayable guard, internal ownership check, storage write, event, and return. Proxy delegatecall preserves the caller and value while targeting proxy storage. No callback, reentrancy edge, or defect was identified. Availability still depends on owner authority, and existing freeze entries remain unchanged.

5.13. Module: FrxUSD3.sol

Function: `permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, byte[32] r, byte[32] s)`

Any relayer may submit a domain-separated permit. Successful authentication consumes the owner's nonce, replaces the allowance, and emits `Approval` without transferring tokens or changing balances or supply. The source declaration uses `bytes32 r` and `bytes32 s`, not the preserved header's `byte[32]` spellings.

Inputs

- `owner`
  - **Control**: Any caller or relayer supplies the owner address.
  - **Constraints**: The owner must authenticate the EIP-712 digest through ECDSA or ERC-1271 and cannot be zero.
  - **Impact**: Selects the nonce, signature authority, allowance owner, event owner, and conditional ERC-1271 target.
- `spender`
  - **Control**: The caller supplies the spender unchanged.

- **Constraints**: The signed spender must be nonzero when approval executes.
- **Impact**: Selects the allowance recipient and `Approval` event spender without receiving tokens or a callback.

- `value`
  - **Control**: The caller supplies any `uint256` value unchanged.
  - **Constraints**: The signed value has no cap, balance check, or nonzero requirement.
  - **Impact**: Replaces the allowance and is emitted without changing balances or supply.
- `deadline`
  - **Control**: The caller supplies the signed deadline unchanged.
  - **Constraints**: The condition `block.timestamp <= deadline` must hold, including equality.
  - **Impact**: Expiry reverts before nonce use, validation, or any ERC-1271 callback.
- `v`
  - **Control**: The caller supplies the `uint8` signature component.
  - **Constraints**: It is appended after `r` and `s` as `r || s || v`.
  - **Impact**: It affects signature acceptance but is not stored or emitted.
- `r`
  - **Control**: The caller supplies the source-ABI `bytes32` component.
  - **Constraints**: It forms the first 32 bytes of the packed signature.
  - **Impact**: It affects ECDSA or ERC-1271 validation only.
- `s`
  - **Control**: The caller supplies the source-ABI `bytes32` component.
  - **Constraints**: It forms bytes 32 through 63 and is subject to the applicable signature rules.
  - **Impact**: It affects signature acceptance but is not stored or emitted.

Branches and code coverage

Intended branches

- Calls and tests use source-ABI `bytes32 r` and `bytes32 s` despite the preserved header.
  - ☑ Test coverage (`TestFrxUSDSignatures.test_Permit_succeeds`)
- The seven-argument wrapper accepts the signed owner, spender, value, deadline, and `v/r/s` tuple and produces the expected allowance.
  - ☑ Test coverage (`TestFrxUSDSignatures.test_Permit_succeeds`)
- The seven-argument path accepts the `r || s || v` EOA signature represented by the

test inputs.

☑ Test coverage (`TestFrxUSDSignatures.test_Permit_succeeds`)

- A call where `block.timestamp < deadline` passes expiry validation.

☑ Test coverage (`TestFrxUSDSignatures.test_Permit_succeeds`)

- A call where `block.timestamp == deadline` also passes expiry validation.

☐ Test coverage

- A valid EOA permit consumes the current nonce and replaces the allowance.

☑ Test coverage (`TestFrxUSDSignatures.test_Permit_succeeds`)

- A successful permit emits `Approval`.

☐ Test coverage

- A contract owner succeeds when ERC-1271 returns the magic value for the exact digest and signature.

☐ Test coverage

- Reusing a successful permit fails because its nonce has been consumed.

☐ Test coverage

- A valid permit with `value == 0` clears the allowance.

☐ Test coverage

- A valid permit with `value == type(uint256).max` stores that allowance.

☐ Test coverage

- Any relayer may submit the permit because `msg.sender` is not authenticated.

☑ Test coverage (`TestFrxUSDSignatures.test_Permit_succeeds`)

- Changing the chain ID or verifying address changes the EIP-712 digest.

☐ Test coverage

Negative behavior

- A call where `block.timestamp > deadline` reverts with `"Permit: expired deadline"` before nonce use or validation.

☐ Negative test

- A malformed, noncanonical, or incorrect-signer EOA signature reverts with `InvalidSignature` and restores state.

☐ Negative test

- A reverting, malformed, or nonmagic ERC-1271 response rejects the permit and restores state.

☐ Negative test

- Changing any authenticated permit or domain field invalidates the signature.

- ☐ Negative test

- Replaying a consumed signature fails without permanently consuming another nonce.
  - ☐ Negative test
- A zero spender reverts during approval after authentication and rolls back the nonce.
  - ☐ Negative test
- A zero owner cannot authenticate and is independently invalid for approval.
  - ☐ Negative test
- A call with nonzero `msg.value` reverts during dispatch before module logic executes.
  - ☐ Negative test
- Any downstream revert atomically restores the nonce, allowance, and logs.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `FrxUSD3.permit` function internally packs `r || s || v`, consumes the nonce, hashes the EIP-712 permit, validates it, and approves. Although the generated result shows only the wrapper, contract-owner validation conditionally makes a `staticcall` to the caller-selected `owner`, which can observe the advanced nonce. Any later revert rolls back all effects. No native value or tokens are transferred, and no receiver callback occurs.

5.14. Module: FrxUSDCustodianBase.sol

Function: `deposit(uint256 _assetsIn, address _receiver)`

This function transfers `_assetsIn` of `custodianTkn` from `msg.sender` into the contract and mints a corresponding amount of frxUSD to `_receiver`, after applying the `mintFee`. The share amount is computed by `previewDeposit`, which deducts the fee before the decimal-scaled conversion.

Inputs

- `_assetsIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxDeposit(_receiver)`.
  - **Impact**: Amount of `custodianTkn` taken from the caller.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.

- **Impact**: Recipient of the deposit.

Branches and code coverage

Intended branches

- In the tested same-address caller/receiver case, the account spends `_assetsIn` of `custodianTkn` and receives `previewDeposit(_assetsIn)` frxUSD within the test's one-unit tolerance.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_deposit`)
- The `Deposit` event is emitted.
  - ☐ Test coverage — event emission is not asserted

Negative behavior

- Reverts with `ERC4626ExceededMaxDeposit` when `_assetsIn > maxDeposit(_receiver)`.
  - ☐ Negative test
- Reverts or enforces a minimum when `_sharesOut` would be zero.
  - ☐ Negative test

Function: `initialize(address _owner, uint256 _mintCap, uint256 _mintFee, uint256 _redeemFee)`

This function initializes the contract when deployed behind a proxy. Because the constructor always sets `wasInitialized = true`, this function can be called only once, and only through the proxy.

Inputs

- `_owner`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets the owner of this contract.
- `_mintCap`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets `mintCap`.
- `_mintFee`

- **Control**: Fully controlled by the caller.
- **Constraints**: None.
- **Impact**: Sets `mintFee`.

- `_redeemFee`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets `redeemFee`.

Branches and code coverage

Intended branches

- Initializes the owner, mint cap, and fees.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_deployment`)

Negative behavior

- Reverts if the contract was already initialized.
  - ☑ Negative test (`FrxUSDCustodianTest.test_deployment`)
- Reverts or warns if `_owner` is `address(0)`.
  - ☐ Negative test — accepted silently

Function: `mint(uint256 _sharesOut, address _receiver)`

This function mints exactly `_sharesOut` frxUSD to `_receiver` by computing the required `custodianTkn` input, which rounds up to favor the protocol.

Inputs

- `_sharesOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxMint(_receiver)`.
  - **Impact**: Exact number of frxUSD shares minted.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Recipient of the minted frxUSD.

Branches and code coverage

Intended branches

- In the tested same-address caller/receiver case, the account pays `previewMint(_sharesOut)` of `custodianTkn` and receives `_sharesOut` of frxUSD within the test's one-unit tolerance.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_mint`)
- The `Deposit` event is emitted.
  - ☐ Test coverage

Negative behavior

- Reverts when `_sharesOut > maxMint(_receiver)`.
  - ☑ Negative test (`FrxUSDCustodianTest.test_mintCap`)
    - The implementation uses `ERC4626ExceededMaxMint`; the test uses generic `expectRevert()` and does not attest the selector.
- Reverts when `_sharesOut` is zero.
  - ☐ Negative test

Function: `recoverERC20(address _tokenAddress, uint256 _tokenAmount)`

This function allows the owner to sweep any ERC-20 token from the contract to the owner address, with no exclusion list.

Inputs

- `_tokenAddress`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No token exclusion list.
  - **Impact**: Determines which token is recovered from the contract.
- `_tokenAmount`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Limited only by the contract's balance of that token.
  - **Impact**: How much is recovered.

Branches and code coverage

Intended branches

- The owner receives the recovered tokens.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_ownerFunctionality`)
- The `RecoveredERC20` event is emitted.
  - ☐ Test coverage — event emission is not asserted

Negative behavior

- Reverts when the caller is not the owner.
  - ☑ Negative test (`FrxUSDCustodianTest.test_ownerFunctionality`)

Function: `redeem(uint256 _sharesIn, address _receiver, address _owner)`

This function burns a specified amount of frxUSD from the caller and releases the corresponding `custodianTkn` (after redeem fee) to `_receiver`.

Inputs

- `_sharesIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxRedeem(_owner)`.
  - **Impact**: Exact number of frxUSD shares burned.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero-address check.
  - **Impact**: Recipient of the `custodianTkn`.
- `_owner`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal `msg.sender`.
  - **Impact**: Source of the frxUSD to be burned.

Branches and code coverage

Intended branches

- In the tested case where caller, receiver, and owner are the same account, it burns `_sharesIn` frxUSD and receives `previewRedeem(_sharesIn)` of `custodianTkn` within the test's one-unit tolerance.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_redeem`)
- The `Withdraw` event is emitted.

- ☐ Test coverage

Negative behavior

- Reverts with `TokenOwnerShouldBeSender` when `_owner != msg.sender`.
  - ☐ Negative test
- Reverts with `ERC4626ExceededMaxRedeem` when `_sharesIn > maxRedeem(_owner)`.
  - ☐ Negative test

Function: `setMintCap(uint256 _mintCap)`

This function updates the `mintCap` that limits cumulative frxUSD minting.

Inputs

- `_mintCap`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Sets `mintCap` that limits cumulative frxUSD minting.

Branches and code coverage

Intended branches

- The `mintCap` is updated.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_ownerFunctionality`)
- The `MintCapSet` event is emitted.
  - ☐ Test coverage — event emission is not asserted

Negative behavior

- Reverts when the caller is not the owner.
  - ☑ Negative test (`FrxUSDCustodianTest.test_ownerFunctionality`)

Function: `setMintRedeemFee(uint256 _mintFee, uint256 _redeemFee)`

This function updates the mint and redeem fees.

Inputs

- `_mintFee`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Must be `< 1e18`.
  - **Impact**: Sets `mintFee`.
- `_redeemFee`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Must be `< 1e18`.
  - **Impact**: Sets `redeemFee`.

Branches and code coverage

Intended branches

- The fees are updated.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_ownerFunctionality`)

Negative behavior

- Reverts when the caller is not the owner.
  - ☑ Negative test (`FrxUSDCustodianTest.test_ownerFunctionality`)
- Reverts when `_mintFee >= 1e18`.
  - ☐ Negative test
- Reverts when `_redeemFee >= 1e18`.
  - ☐ Negative test

Function: `withdraw(uint256 _assetsOut, address _receiver, address _owner)`

This function burns the caller's frxUSD (computed from `_assetsOut` via `previewWithdraw`) and releases `_assetsOut` of `custodianTkn` to `_receiver`.

Inputs

- `_assetsOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxWithdraw(_owner)`.
  - **Impact**: Amount of `custodianTkn` received by `_receiver`.

- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero-address check.
  - **Impact**: Recipient of the `custodianTkn`.
- `_owner`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal `msg.sender`; otherwise, the function reverts with `TokenOwnerShouldBeSender`.
  - **Impact**: Source of the frxUSD to be burned.

Branches and code coverage

Intended branches

- In the tested case where caller, receiver, and owner are the same account, it burns `previewWithdraw(_assetsOut)` frxUSD and receives `_assetsOut` of `custodianTkn` within the test's one-unit tolerance.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_withdraw`)
- The `Withdraw` event is emitted.
  - ☐ Test coverage

Negative behavior

- Reverts with `TokenOwnerShouldBeSender` when `_owner != msg.sender`.
  - ☐ Negative test
- Reverts when `_assetsOut > maxWithdraw(_owner)`.
  - ☑ Negative test (`FrxUSDCustodianTest.test_withdrawCap`)
    - The implementation uses `ERC4626ExceededMaxWithdraw`; the test uses generic `expectRevert()` and does not attest the selector.

5.15. Module: FrxUSDCustodianUsdc.sol

Function: `redeemWtgxx(uint256 amountWtgxx)`

This function is the entry point for the async WTGXX-to-USDC redemption flow. It is called by `FrxUSDCustodianWithReceiver.redeemForUsdcAsync`, which first transfers WTGXX tokens into this contract and then calls this function to forward them to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`, which then processes the on-chain receipt and sends back USDC.

Inputs

- `amountWtgxx`
  - **Control**: Fully controlled by the caller (`WTGXX_CUSTODIAN_MAINNET`).
  - **Constraints**: Must not exceed the WTGXX balance of this contract; otherwise, the underlying `transfer` reverts.
  - **Impact**: The amount of WTGXX forwarded to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`.

Branches and code coverage

Intended branches

- WTGXX tokens are successfully transferred to `ON_RECEIPT_PRIMARY_ORDER_ADDRESS`.
  - ☐ Test coverage
- The `WtgxxRedemptionAsync` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function reverts if `msg.sender` is not `WTGXX_CUSTODIAN_MAINNET`.
  - ☐ Negative test
- The function reverts if the contract's WTGXX balance is less than `amountWtgxx`.
  - ☐ Negative test

5.16. Module: FrxUSDCustodianUsdcV2.sol

Function: `_isFeeExempt()`

This function returns `true` if either `msg.sender` or `tx.origin` appears in the `feeExempt` mapping. All four `preview*` overrides gate on this.

Branches and code coverage

Intended branches

- Returns `true` when `msg.sender` is in `feeExempt`.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_previews_are_fee_free_when_sende`
- Returns `true` when `tx.origin` is in `feeExempt` and `msg.sender` is not.
  - ☑ Test coverage

(FrxUSDCustodianUsdcV2UpgradeTest.test_exempt_via_origin_through_interm

- Returns `false` when neither address is exempt.

  ☑ Test coverage (FrxUSDCustodianUsdcV2UpgradeTest.test_previews_charge_fees_when_not_ex

Function: `previewDeposit(uint256 _assetsIn)`

This function overrides the base `previewDeposit`. It applies `mintFee` only when `_isFeeExempt()` returns `false`.

Inputs

- `_assetsIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No explicit zero check.
  - **Impact**: Determines how many shares are minted.

Branches and code coverage

Intended branches

- Fee-exempt caller: the fee is zero, and the full `_assetsIn` converts to shares.

  ☑ Test coverage (FrxUSDCustodianUsdcV2UpgradeTest.test_previews_are_fee_free_when_sende FrxUSDCustodianUsdcV2UpgradeTest.test_deposit_exempt)

- Nonexempt caller with nonzero `mintFee`: `_assetsIn` is reduced by the fee fraction before conversion.

  ☑ Test coverage (FrxUSDCustodianUsdcV2UpgradeTest.test_previews_charge_fees_when_not_ex FrxUSDCustodianUsdcV2UpgradeTest.test_deposit_not_exempt)

- Nonexempt caller with `mintFee == 0`: the result is the same as on the fee-exempt path.

  ☐ Test coverage

Function: `previewMint(uint256 _sharesOut)`

This function overrides the base `previewMint`. It applies `mintFee` only when `_isFeeExempt()` returns `false`.

Inputs

- `_sharesOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Target number of shares — determines the assets that must be supplied.

Branches and code coverage

Intended branches

- Fee-exempt caller: no fee markup is applied to assets.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_mint_exempt`)
- Nonexempt caller: `_assetsIn` is scaled up by `1e18 / (1e18 - fee)` (ceil).
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_previews_charge_fees_when_not_ex`

Function: `previewRedeem(uint256 _sharesIn)`

This function overrides the base `previewRedeem`. It applies `redeemFee` to reduce `_assetsOut` after converting shares to assets.

Inputs

- `_sharesIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Number of shares to redeem.

Branches and code coverage

Intended branches

- Fee-exempt caller: the full asset value is returned.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_redeem_exempt`)
- Nonexempt caller: `_assetsOut = (1e18 - redeemFee) * _assetsOut / 1e18` (floor).
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_redeem_not_exempt`)

Function: `previewWithdraw(uint256 _assetsOut)`

This function overrides the base `previewWithdraw`. It applies `redeemFee` to gross up `_assetsOut`.

Inputs

- `_assetsOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Desired net asset withdrawal — determines shares required.

Branches and code coverage

Intended branches

- Fee-exempt caller: no gross-up is applied, and shares are calculated directly from `_assetsOut`.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_withdraw_exempt`)
- Nonexempt caller: `_assetsOut` is grossed up by `1e18 / (1e18 - redeemFee)` (ceil) before share conversion.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_previews_charge_fees_when_not_ex`

Function: `setFeeExempt(address _user, bool _isExemptFromFee)`

This is an owner-only function that adds or removes an address from the `feeExempt` mapping.

Inputs

- `_user`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No zero-address check.
  - **Impact**: The address granted or revoked fee exemption.
- `_isExemptFromFee`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Enables or disables the fee for that address across all deposit, mint, withdraw, and redeem flows.

Branches and code coverage

Intended branches

- Grants fee exemption and emits the event.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_setFeeExempt`)
- Revokes fee exemption and emits the event.
  - ☑ Test coverage (`FrxUSDCustodianUsdcV2UpgradeTest.test_setFeeExempt`, `FrxUSDCustodianUsdcV2UpgradeTest.test_revoked_exemption_charges_fees_again`)

Negative behavior

- Reverts when called by a non-owner.
  - ☑ Negative test (`FrxUSDCustodianUsdcV2UpgradeTest.test_setFeeExempt_onlyOwner`)

5.17. Module: FrxUSDCustodianWithOracle.sol

Function: `_convertToAssets(uint256 _shares, Math.Rounding _rounding)`

This function is the inverse of `_convertToShares`. It multiplies the base decimal-normalized asset value by `10 ** oracleDecimals` and divides by `getCustodianOraclePrice()`.

Inputs

- `_shares`
  - **Control**: Propagated from callers.
  - **Constraints**: None.
  - **Impact**: Share count converted to assets at the oracle-adjusted rate.
- `_rounding`
  - **Control**: Hardcoded by callers.
  - **Constraints**: None.
  - **Impact**: Determines rounding bias.

Branches and code coverage

Intended branches

- The oracle price divides correctly for a standard redeem.
  - ☑ Test coverage (`FrxUSDCustodianWithOracleTest.test_redeem`)

- A non-unit oracle price changes the custodian-token output relative to a 1:1 conversion.
  - ☐ Test coverage — `FrxUSDCustodianWithOracleTest.test_redeem` uses the fetched price in its expected-value formula but does not assert that the price is non-unit or compare the output with a 1:1 baseline

Negative behavior

- An oracle price much larger than `10 ** oracleDecimals` can cause the `mulDiv` to floor-truncate to `0`, releasing zero assets while still burning the caller's shares.
  - ☐ Negative test

Function: `_convertToShares(uint256 _assets, Math.Rounding _rounding)`

This function overrides the base 1:1 decimal-adjusted conversion by multiplying the result by `getCustodianOraclePrice()` and dividing by `10 ** oracleDecimals`.

Inputs

- `_assets`
  - **Control**: Propagated from callers.
  - **Constraints**: None.
  - **Impact**: Asset amount converted to shares at the oracle-adjusted rate.
- `_rounding`
  - **Control**: Hardcoded by callers.
  - **Constraints**: N/A.
  - **Impact**: Determines rounding bias (`Floor` for deposits, `Ceil` for withdrawals) applied consistently across both the inner and outer `mulDiv`.

Branches and code coverage

Intended branches

- The oracle price is applied correctly for a standard deposit.
  - ☑ Test coverage (`FrxUSDCustodianWithOracleTest.test_deposit`)
- A non-unit oracle price changes the share output relative to a 1:1 conversion.
  - ☐ Test coverage — `FrxUSDCustodianWithOracleTest.test_deposit` uses the fetched price in its expected-value formula but does not assert that the price is non-unit or compare the output with a 1:1 baseline

Negative behavior

- Dust deposits where `super._convertToShares` returns `0` silently yield zero shares, consuming assets with no output.
  - ☐ Negative test
- An `oracleDecimals` mismatch after `setCustodianOracle` in the same block produces incorrect share amounts.
  - ☐ Negative test

Function: `getCustodianOraclePrice()`

This function returns the oracle price for the custodian token. It uses a block-level cache: if `lastOracleUpdate == block.number`, return `lastSavedOraclePrice` without querying Chainlink.

Branches and code coverage

Intended branches

- Returns the cached price if the oracle was already fetched this block.
  - ☐ Test coverage — no scoped test independently asserts cache use
- Fetches and validates a fresh Chainlink price when no cache exists for the current block.
  - ☐ Test coverage — no scoped test independently asserts the fresh-read result

Negative behavior

- Reverts with `OracleError` if the Chainlink answer is `<= 0`.
  - ☐ Negative test
- Reverts with `OracleError` if the price is stale (`block.timestamp - updatedAt > maximumOracleDelay`).
  - ☐ Negative test

Function: `initialize(address _owner, address _custodianOracle, uint256 _maximumOracleDelay, uint256 _mintCap, uint256 _mintFee, uint256 _redeemFee)`

This function initializes the proxy instance and sets oracle-specific state. It can only be called once, which `wasInitialized` in the base class enforces.

Inputs

- `_owner`

- **Control**: Fully controlled by the caller.
- **Constraints**: Must not be the zero address.
- **Impact**: Becomes the owner with full privileged access over the contract.

- `_custodianOracle`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No explicit zero-address check.
  - **Impact**: Sets `custodianOracle` and `oracleDecimals` used in all subsequent price conversions.
- `_maximumOracleDelay`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No lower or upper bound.
  - **Impact**: Sets the maximum allowed age (in seconds) of a Chainlink price before the contract treats it as stale.
- `_mintFee`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets `mintFee`, the proportion of the deposit that is retained as a fee.
- `_redeemFee`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Same missing validation as `_mintFee`.
  - **Impact**: Sets `redeemFee`, the proportion of the withdrawal that is retained as a fee.
- `_mintCap`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No bounds.
  - **Impact**: Sets `mintCap`, the maximum total frxUSD the contract may mint.

Branches and code coverage

Intended branches

- The contract is marked initialized, and the owner is stored.
  - ☑ Test coverage (`FrxUSDCustodianWithOracleTest.test_deployment`)
- The oracle address, decimals, and maximum delay are stored correctly.
  - ☐ Test coverage — oracle parameters are not asserted
- Fees and mint cap are stored correctly.
  - ☑ Test coverage (`FrxUSDCustodianWithOracleTest.test_deployment`)

Negative behavior

- The function reverts if already initialized.
  - ☑ Negative test (`FrxUSDCustodianWithOracleTest.test_deployment`)
- The function reverts if `_owner == address(0)`.
  - ☐ Negative test

Function: `setCustodianOracle(address _custodianOracle, uint256 _maximumOracleDelay)`

This is an owner-gated function to replace the custodian oracle address and its maximum staleness delay.

Inputs

- `_custodianOracle`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Updates `custodianOracle` and refreshes `oracleDecimals` to match the new feed.
- `_maximumOracleDelay`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No lower or upper bound.
  - **Impact**: Replaces the maximum allowed age of a Chainlink price.

Branches and code coverage

Intended branches

- The oracle address and delay are updated successfully.
  - ☐ Test coverage
- The stored `oracleDecimals` value is refreshed to match the new oracle.
  - ☐ Test coverage

Negative behavior

- The function reverts if called by a non-owner.
  - ☐ Negative test
- When the new oracle has different decimals and a same-block cache already exists (`lastOracleUpdate == block.number`), subsequent same-block calls use the old

cached price magnitude with the new `oracleDecimals` divisor, causing mispricing.

- ☐ Negative test

5.18. Module: FrxUSDOFTUpgradeable.sol

Function: `addFreezer(address account)`

The `addFreezer` function, callable only by the owner, dispatches through `FreezeThawModule._addFreezer` to the linked library. This updates namespaced freezer storage and emits indexed `AddFreezer(account)`. A callable freezer or the owner can later freeze the zero address, blocking non-owner mint or burn paths. Deployment state is unspecified.

Inputs

- `account`
  - **Control**: The owner selects any ABI-valid address, including zero, while `msg.sender` must be the current owner and `msg.value` must be zero.
  - **Constraints**: The address must not already be present, or `FreezeThawLib.addFreezer` reverts with `AlreadyFreezer()`.
  - **Impact**: Successful insertion updates the current execution context's namespaced freezer set and emits indexed `AddFreezer(account)` without moving tokens or native value.

Branches and code coverage

Intended branches

- The owner inserts a new tested account, which becomes visible through `isFreezer` and `freezers()`.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_AddFreezer_succeeds`)
- Success emits `AddFreezer(account)` once.
  - ☐ Test coverage
- A new `address(0)` account is accepted and stored, and `AddFreezer(address(0))` is emitted. The entry is visible through `isFreezer(address(0))` and `freezers()`. The zero address cannot originate a call.
  - ☐ Test coverage

Negative behavior

- A non-owner caller is rejected.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_AddFreezer_OnlyOwner`)

- Reversion before library dispatch, mutation, or event emission is not instrumented.
  - ☐ Negative test
- Adding an existing freezer reverts with `AlreadyFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_AddFreezer_AlreadyFreezer_reverts`)
- Preserved membership and absence of an event are not independently asserted.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point before function execution.
  - ☐ Negative test
- Short or noncanonical calldata reverts during ABI decoding before authorization or mutation.
  - ☐ Negative test
- A linked-library revert or failed `DELEGATECALL` propagates and rolls back all storage changes and logs.
  - ☐ Negative test

Function call analysis

- `this._addFreezer(account) -> FreezeThawLib.addFreezer(account)`
  - The trace misrenders the internal `_addFreezer(account)` dispatch. It resolves to `FreezeThawModule._addFreezer`, after which the fixed-linked `FreezeThawLib.addFreezer` executes via `DELEGATECALL` in the current or proxy context, without callbacks or a reentrancy point.
- **Manual call-path verification**: The library mutates namespaced freezer storage and emits from the current context. It reverts if the account already exists. Parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` affect only sibling `_quote` paths.

Function: `burn(address account, uint256 amount)`

This owner-only administrative burn resolves a zero amount to the account's full balance. The inherited `_burn` function reduces balance and total supply, runs hooks, and emits `Transfer`. Calling `_burn` directly bypasses allowances, rate limits, and supply tracking, while internal dispatch preserves the owner as `msg.sender` and bypasses owner-path compliance checks.

Inputs

- `account`

  - **Control**: The current owner selects the burn source; `tx.origin`, signatures, allowances, and external callees are unused.
  - **Constraints**: The zero address reverts in `_burn`, while renounced ownership makes the entry point unreachable.
  - **Impact**: Its balance is reduced and it becomes the `from` address for hooks and `Transfer`, without receiving a callback.
- `amount`
  - **Control**: The current owner selects the amount without oracle, bridge, rate-limit, or external-return input.
  - **Constraints**: Zero resolves to `balanceOf(account)`, while a nonzero value remains unchanged and cannot exceed that balance.
  - **Impact**: The resolved value determines the balance and total-supply decrease, with a zero burn still running hooks and emitting `Transfer`.

Branches and code coverage

Intended branches

- Calls from the current owner pass authorization and complete partial and full burns.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Burn_PartialAmount_succeeds`, `FrxUSDOFTUpgradeableTest.test_Burn_FullAmount_succeeds`)
- A differing `tx.origin` is not exercised.
  - ☐ Test coverage
- When `amount == 0`, the function reads `balanceOf(account)` and burns that full stored balance.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Burn_FullAmount_succeeds`)
- When `amount != 0`, the supplied value is passed unchanged to `_burn`.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Burn_PartialAmount_succeeds`)
- Because internal dispatch preserves the authorized owner as `msg.sender`, `_beforeTokenTransfer` skips pause and freeze reads before calling its inherited hook.
  - ☐ Test coverage
- A valid partial burn decreases the tested account balance by the exact amount.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Burn_PartialAmount_succeeds`)
- The burn also decreases supply, emits `Transfer`, runs the after-transfer hook, and returns no data.

- ☐ Test coverage

- For a nonzero zero-balance account and `amount == 0`, a zero burn succeeds without numeric changes but still emits the zero-value `Transfer`.
  - ☐ Test coverage

Negative behavior

- A non-owner caller reverts.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Burn_OnlyOwner`)
- Ordering before balance reads and effects is not independently asserted.
  - ☐ Negative test
- A call where `account == address(0)` reverts in `_burn` before hooks, balance or supply changes, and event emission.
  - ☐ Negative test
- A resolved nonzero amount above the account balance reverts after the owner-exempt pretransfer hook, rolling back all effects and logs.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable wrapper before authorization or function-body execution.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The scoped path is `burn` → `onlyOwner` / `owner()` → optional internal `balanceOf(account)` → inherited `_burn` → FrxUSDOFTUpgradeable `_beforeTokenTransfer` → inherited hooks and ERC-20 effects. The authorized owner remains `msg.sender`, so freeze checks are skipped. Unresolved sibling `_quote` calls at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` are unreachable.

Function: `burnMany(address[] accounts, uint256[] amounts)`

This owner-only batch burn processes equal-length arrays sequentially. Zero amounts burn each account's then-current balance, so duplicates observe prior burns. The owner bypasses pause and freeze checks.

Inputs

- `accounts`
  - **Control**: The owner selects each burn source.

    - **Constraints**: Length must match `amounts`; nonzero addresses and duplicates are allowed.
    - **Impact**: Each address identifies the balance reduced and the `Transfer` event's `from`.
- `amounts`
    - **Control**: The owner selects each requested amount.
    - **Constraints**: Nonzero values burn exactly, while zero means the account's full current balance.
    - **Impact**: Each resolved value reduces the selected balance and total supply equally.
- `msg.sender`
    - **Control**: The EVM call context sets the caller.
    - **Constraints**: It must equal current `owner()`, including when the owner is a contract.
    - **Impact**: It authorizes the batch and selects the transfer hook's owner-bypass path.
- `msg.value`
    - **Control**: The caller sets native value at the call boundary.
    - **Constraints**: The nonpayable dispatcher requires zero value.
    - **Impact**: No native value is accepted, retained, or forwarded.
- `tx.origin`
    - **Control**: The originating EOA determines this value.
    - **Constraints**: The resolved execution path never reads it.
    - **Impact**: It has no authorization or state effect.

Branches and code coverage

Intended branches

- An owner call with equal nonempty arrays enters the loop; each account must be nonzero, and each explicit amount must not exceed its then-current balance.
    - ☐ Test coverage
- Equal empty arrays return without state changes or logs.
    - ☐ Test coverage
- A nonzero amount burns that exact value.
    - ☐ Test coverage
- A zero amount burns the account's then-current full balance.
    - ☐ Test coverage

- A zero sentinel on an empty nonzero account performs a zero burn and emits a zero-value `Transfer`.
  - ☐ Test coverage
- Multiple entries execute sequentially, so duplicate accounts observe prior burns.
  - ☐ Test coverage
- Owner-authorized burns bypass pause and freeze evaluation while still reaching inherited hooks; freezing `address(0)` blocks non-owner mint and burn routes but not `burnMany`.
  - ☐ Test coverage

Negative behavior

- A caller other than `owner()` reverts.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_BurnMany_OnlyOwner`)
- Ordering before length checks and burns is not instrumented.
  - ☐ Negative test
- Unequal array lengths revert with `ArrayMisMatch()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_BurnMany_ArrayMismatch_reverts`)
- Ordering before loop entry is not instrumented.
  - ☐ Negative test
- Burning from `address(0)` reverts and rolls back the batch.
  - ☐ Negative test
- A nonzero amount above the account's current balance reverts atomically.
  - ☐ Negative test
- A duplicate account can fail when an earlier burn leaves insufficient balance for a later explicit amount.
  - ☐ Negative test
- Any later revert or out-of-gas failure rolls back all prior balance, supply, and log effects.
  - ☐ Negative test
- Nonzero `msg.value` is rejected before function-body execution.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The path is dispatcher → `onlyOwner` / `owner()` → length check → optional `balanceOf()` → `_burn()` → owner-bypassed `_beforeTokenTransfer()`

/ super → ERC-20 accounting and `Transfer` → `_afterTokenTransfer()`, with no external interaction. The generated FCA omits these internal/inherited steps, and ASI failures at `contracts/FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `contracts/FraxOFTUpgradeableTempo.sol@48:2180` concern unrelated sibling Tempo paths.

Function: `checkpointRateLimits(uint32[] _eids)`

This owner-only wrapper forwards `_eids` unchanged through an internal helper to a linked RateLimiterLib via fixed-target `DELEGATECALL`. It updates caller/proxy namespaced storage and emits `RateLimitCheckpointed` for each occurrence.

Inputs

- `_eids`
  - **Control**: The authorized owner controls the ordered `uint32` array, including its length, duplicates, and omissions.
  - **Constraints**: Empty arrays are accepted; no uniqueness, sorting, configuration, or explicit length checks apply; and nonzero `msg.value` is rejected.
  - **Impact**: Each occurrence checkpoints its EID using override or default configuration, stored state, and `uint32(block.timestamp)`; duplicates repeat, while omitted EIDs remain untouched.

Branches and code coverage

Intended branches

- An authorized caller forwards the complete `_eids` array internally to the fixed linked-library delegatecall.
  - ☐ Test coverage
- An empty array completes without per-EID writes or checkpoint events.
  - ☐ Test coverage
- Each EID uses its override configuration when enabled and otherwise uses `defaultConfig`.
  - ☐ Test coverage
- If `lastUpdated == 0` or `lastUpdated >= uint32(block.timestamp)`, usage is preserved and the shared timestamp is updated.
  - ☐ Test coverage
- If `0 < lastUpdated < uint32(block.timestamp)`, both directions decay

independently using the same elapsed interval.

- ☐ Test coverage

- Each direction preserves usage for zero usage, limit, or window; otherwise, it clears after a full window or subtracts bounded floored recovery.
  - ☐ Test coverage
- Duplicate EIDs are processed and logged repeatedly, but same-block repetitions apply no additional decay.
  - ☐ Test coverage
- Global and directional enabled flags are ignored, so inactive nonzero parameters may decay usage while zero limit or window preserves it.
  - ☐ Test coverage
- The shared timestamp couples both directions, and positive-time checkpoints may floor subunit recovery.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts before the function body, producing no checkpoint writes or events.
  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable entry point.
  - ☐ Negative test
- Malformed `uint32[]` calldata reverts during dispatch or decoding before mutation.
  - ☐ Negative test
- Delegatecall failure, revert, panic, or out-of-gas atomically rolls back all batch writes and logs.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The `this._checkpointRateLimits` prefix is inaccurate because the helper call is internal; only the linked-library edge is a fixed-target `DELEGATECALL`, which preserves caller context, returns no application data, and makes no further calls or value transfers.
- **Manual call-path verification**: The `checkpointRateLimits` function calls the inherited `_checkpointRateLimits` helper internally, which delegates to `RateLimiterLib.checkpointRateLimits`. Failures propagate atomically, and the

linked target is a deployment assumption.

Function: `constructor(address _lzEndpoint)`

Deployment forwards the selected endpoint unchanged to OFTUpgradeable then disables initializers in implementation-local storage. Proxy storage remains separate and requires external initialization.

Inputs

- `_lzEndpoint`
  - **Control**: The creator selects the endpoint address.
  - **Constraints**: The target performs no nonzero, code, interface, identity, or allowlist validation; imported-base validation is unknown.
  - **Impact**: The value is forwarded unchanged to OFTUpgradeable and determines the implementation's endpoint configuration.
- `msg.sender`
  - **Control**: The immediate creator is an EOA or deployment contract.
  - **Constraints**: Target constructor logic does not read or forward `msg.sender`.
  - **Impact**: It has no target-defined effect on control flow, ownership, or initialization.
- `msg.value`
  - **Control**: The creator selects the native value attached to deployment.
  - **Constraints**: Because the constructor is nonpayable, successful deployment requires `msg.value == 0`.
  - **Impact**: Nonzero value reverts creation, leaving no runtime code, state, or logs.
- `tx.origin`
  - **Control**: The transaction-originating EOA determines this value.
  - **Constraints**: Target constructor logic does not read or forward `tx.origin`.
  - **Impact**: It has no target-defined effect on construction or state.
- Inherited initializer state at `_disableInitializers()` entry
  - **Control**: Fresh implementation storage and preceding base-constructor execution determine this state.
  - **Constraints**: The `_disableInitializers()` function locks an unlocked state, makes no transition if already disabled, and reverts during active initialization.
  - **Impact**: A successful transition locks implementation-local initialization without locking separate proxy storage.

Branches and code coverage

Intended branches

- With zero value, successful `OFTUpgradeable(_lzEndpoint)` construction reaches the target constructor body.
  - ☐ Test coverage — no supplied test result is mapped to this exact target-constructor path
- If initializer state is neither disabled nor active, `_disableInitializers()` applies the terminal implementation lock.
  - ☐ Test coverage — no supplied test is mapped to this implementation-lock transition or its event
- If initializer state is already disabled, `_disableInitializers()` makes no new transition.
  - ☐ Test coverage — no supplied test establishes this exact constructor-state branch
- Successful completion installs runtime code and returns the implementation address through the creation operation.
  - ☐ Test coverage — no supplied evidence maps to the complete successful deployment state for this target declaration

Negative behavior

- Nonzero `msg.value` reverts the nonpayable creation and rolls back all creation-frame effects.
  - ☐ Negative test — no exact nonpayable-constructor test was supplied for FrxUSDOFTUpgradeable
- A base-constructor revert propagates before target construction completes.
  - ☐ Negative test — no exact imported-base-constructor failure test was supplied for this target
- Active initialization causes `_disableInitializers()` to revert and abort creation.
  - ☐ Negative test — no exact inherited-helper failure test was supplied
- Invalid endpoint values receive no target-local rejection, while imported-base acceptance remains unknown.
  - ☐ Negative test — no exact invalid-endpoint deployment test was supplied for this target

Function call analysis

- Call path: `constructor(address) -> OFTUpgradeable(_lzEndpoint)`

    - This statically bound base construction receives `_lzEndpoint` unchanged; its unavailable imported implementation leaves nested checks, state changes, events, and external interactions undetermined.
- Call path: `constructor(address) -> _disableInitializers()`
    - This inherited internal call locks implementation-local initializer state, makes no transition if already disabled, or reverts during active initialization.
- Creation boundary: `creator -> CREATE/CREATE2 -> constructor(address)`
    - The creator selects the argument and creation mechanism; success installs runtime code, while any revert rolls back the complete creation frame.
- **Manual call-path verification**: ASI emitted no generated FCA sink header, so manual tracing recovered these two calls and the outer creation boundary. Sibling `_quote` parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` do not affect this constructor.

Function: `freeze(address account)`

Authorized freezers or the owner can add any caller-selected address to the frozen set through a linked-library delegatecall and emit `AccountFrozen`. Duplicate and zero addresses succeed. The function returns no data and transfers no tokens or native value.

Inputs

- `account`
    - **Control**: The caller selects the address forwarded unchanged to the library.
    - **Constraints**: Any ABI-valid address, including zero or an already frozen address, is accepted.
    - **Impact**: The address becomes the frozen-set key and the `AccountFrozen(account)` event argument.
- `msg.sender` (implicit)
    - **Control**: The immediate EVM caller determines this value.
    - **Constraints**: The caller must be a configured freezer or the current owner.
    - **Impact**: It controls authorization but is neither stored nor emitted.
- `msg.value` (implicit)
    - **Control**: The external caller selects the transaction value.
    - **Constraints**: The nonpayable entry point permits only zero value.
    - **Impact**: It is otherwise unused, and the delegatecall transfers no native value.
- `tx.origin` (implicit)
    - **Control**: The transaction originator determines this value.

- **Constraints**: This path never reads or compares it.
- **Impact**: It has no effect on authorization, storage, events, or dispatch.

Branches and code coverage

Intended branches

- A configured non-owner freezer reaches `_freeze(account)`.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeAsFreezer_succeeds`)
- Owner fallback authorization: a non-freezer owner passes the conditional owner comparison and reaches `_freeze(account)`.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeAsOwner_succeeds`)
- A fresh tested address is inserted into frozen membership.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeAsFreezer_succeeds`, `FrxUSDOFTUpgradeableTest.test_FreezeAsOwner_succeeds`)
- A fresh freeze emits `AccountFrozen(account)`.
  - ☐ Test coverage
- Idempotent repeated freeze: a duplicate add returns false, but the ignored result allows success and another event.
  - ☐ Test coverage
- Zero-address freeze: freezing zero succeeds, blocking non-owner mint and burn hooks until the owner thaws zero, while owner-initiated token operations retain their bypass.
  - ☐ Test coverage

Negative behavior

- A caller that is neither freezer nor owner receives `NotFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Freeze_NotFreezer_reverts`)
- Reversion before delegatecall or mutation is not instrumented.
  - ☐ Negative test
- Nonpayable boundary: nonzero `msg.value` is rejected before the function body executes.
  - ☐ Negative test
- Authorization-read failure: any exceptional authorization-read failure propagates before state changes or event emission.

- ☐ Negative test

- Linked-library failure: a failed or reverting delegatecall propagates and atomically rolls back the insertion and event.
  - ☐ Negative test

Function call analysis

- `this.isFreezer(msg.sender) -> this._getFreezeThawStorage()`
  - This is internal dispatch to a fixed namespaced storage slot, and the assembly performs no external call.
- `this.isFreezer(msg.sender) -> EnumerableSet.contains($.freezers, account)`
  - The inner `account` is the outer `msg.sender`, and `contains` only reads freezer membership and cannot reenter.
- `this._freeze(account) -> FreezeThawLib.freeze(account)`
  - The `_freeze` function internally reaches a fixed linked-library `DELEGATECALL` that preserves calling context, adds the address, emits the event, exposes no callback, and depends on trusted linking and library bytecode.
- **Manual call-path verification**: ASI's `this.*` labels misstate internal dispatch and omit the conditional `owner()` lookup, delegatecall semantics, `EnumerableSet.add`, and event emission. The verified path is authorization, `_freeze(account)`, fixed-target library delegatecall, set update, and emission.

Function: `freezeMany(address[] accounts)`

Authorized freezers or the owner delegate `accounts` to the linked library, which updates frozen membership at `0xf6192aebe04228448ed4fd2802d59c009f5324cdd6a110c65bd318373b6e1d00` and emits `AccountFrozen` per occurrence. Zero addresses and duplicates are accepted.

Inputs

- `accounts`
  - **Control**: The caller controls the array length, order, duplicates, and every address.
  - **Constraints**: ABI decoding and authorization are required, but no nonzero, uniqueness, ordering, or explicit batch-size restriction applies.
  - **Impact**: Distinct addresses become frozen while every occurrence emits an event, and any failure rolls back the complete batch and logs.

Branches and code coverage

Intended branches

- A configured non-owner freezer processes the complete tested array.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeManyAsFreezer_succeeds`)
- If freezer membership is false but `msg.sender == owner()`, the owner is authorized and the complete array is processed.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeManyAsOwner_succeeds`)
- If `accounts.length == 0`, the call succeeds without storage changes or events.
  - ☐ Test coverage
- A nonempty tested array inserts both elements into frozen membership.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_FreezeManyAsFreezer_succeeds`, `FrxUSDOFTUpgradeableTest.test_FreezeManyAsOwner_succeeds`)
- Each occurrence emits `AccountFrozen` in order.
  - ☐ Test coverage
- Duplicate or already frozen addresses leave membership deduplicated but still emit an event per occurrence.
  - ☐ Test coverage
- Supplying `address(0)` freezes zero, blocking non-owner OFT debit and credit because transfer checks inspect burn `to` and mint `from`.
  - ☐ Test coverage
- If the linked-library call and all iterations complete, every update and log persists atomically.
  - ☐ Test coverage

Negative behavior

- A caller that is neither freezer nor owner reverts with `NotFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_FreezeMany_NotFreezer_reverts`)
- Reversion before all effects is not independently asserted.
  - ☐ Negative test
- Malformed `address[]` calldata is rejected before the function body and authorization.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by Solidity's nonpayable entry machinery.

- ☐ Negative test

- Delegatecall failure, panic, revert, or out-of-gas rolls back the entire batch and all logs.
  - ☐ Negative test

Function call analysis

- `this.isFreezer(msg.sender) -> this._getFreezeThawStorage()`
  - ASI mislabels this internal path; the helper only returns the fixed namespaced storage pointer and performs no external call.
- `this.isFreezer(msg.sender) -> EnumerableSet.contains($.freezers, account)`
  - This internal storage lookup checks whether `msg.sender` belongs to the owner-managed freezer set and cannot call back or reenter.
- `this._freezeMany(accounts) -> FreezeThawLib.freezeMany(accounts)`
  - The `_freezeMany` function resolves internally to `FreezeThawModule._freezeMany`, whose compiler-generated `DELEGATECALL` executes the linked library in the caller's storage and identity context without source-level callbacks.
- **Manual call-path verification**: ABI decoding and nonpayability precede authorization. Authorized calls reach the linked-library delegatecall, whose source only updates the frozen set and emits events. ASI omits the conditional owner read and the delegatecall/storage/event details, and it reports unrelated builder errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180`.

Function: `initialize(address _delegate)`

This ungated `reinitializer(3)` function initializes proxy ERC-20/OFT/OApp and EIP-712 state, sets `_delegate` on the fixed endpoint, and transfers ownership to `_delegate`. Initialization access must be protected for eligible proxies. It changes no supply, balances, rate limits, freeze, pause, bridge, or fee state.

Inputs

- `_delegate`
  - **Control**: The successful external caller selects `_delegate`, while the endpoint address is constructor-fixed.
  - **Constraints**: No local validation rejects zero or requires `_delegate == msg.sender`; execution depends on version eligibility and endpoint acceptance.

- **Impact**: The `_delegate` parameter is forwarded unchanged to `endpoint.setDelegate` and stored as the final owner.

Branches and code coverage

Intended branches

- An inactive proxy below version 3 records version 3, enters initialization, executes the body, and clears the active flag on success.
  - ☐ Test coverage
- The `__OFT_init("Frax USD", "frxUSD", _delegate)` call initializes ERC-20/OFT/OApp state, sets ownership to `msg.sender`, and forwards `_delegate` to the endpoint.
  - ☐ Test coverage
- After the endpoint returns, `__EIP712_init("Frax USD", "1.2.0")` writes domain strings and clears legacy hashes in fixed storage.
  - ☐ Test coverage
- The explicit `__Ownable_init()` resets ownership to `msg.sender`, and then `_transferOwnership(_delegate)` stores the final owner and returns no data.
  - ☐ Test coverage

Negative behavior

- Version 3 or higher, active initialization, or the implementation's initializer lock causes `reinitializer(3)` to revert before the body.
  - ☐ Negative test
- No owner, role, or signature gate prevents an arbitrary caller from initializing an eligible proxy and selecting its delegate and owner.
  - ☐ Negative test
- The nonpayable ABI rejects nonzero `msg.value`, and the resolved path does not read `tx.origin`.
  - ☐ Negative test
- Zero `_delegate` succeeds only if the endpoint accepts it, after which final ownership is also zero.
  - ☐ Negative test
- An endpoint revert rolls back versioning, metadata/OFT initialization, ownership changes, and endpoint effects.
  - ☐ Negative test
- Any later revert or out-of-gas condition also atomically rolls back local and endpoint state.

- ☐ Negative test

- No supplied test is mapped to these initializer branches, so all coverage markers remain unchecked.
  - ☐ Negative test

Function call analysis

- `this.__OFT_init(this.name(), this.symbol(), _delegate) -> this.__OFTCore_init(_delegate) -> this.__OAppCore_init(_delegate) -> this.__OAppCore_init_unchained(_delegate) -> this.endpoint.setDelegate(_delegate)`
  - The generated `this.*` calls are internal; the sole external edge is the trusted, constructor-fixed `endpoint.setDelegate(_delegate)`. The call sends zero value, consumes no return data, and propagates reverts. During callbacks, version 3 is observable and the stored interim owner remains the initializer caller. Callback `msg.sender` is the endpoint and matches that owner only if both addresses are equal.
- `this.__EIP712_init(this.name(), this.version()) -> this.__EIP712_init_unchained(name, version) -> this._getEIP712Storage()`
  - Manual review shows only fixed-slot pointer assignment and storage writes. The internal helper is unguarded, but in this path it executes inside `reinitializer(3)` and is not externally exposed.
- **Manual call-path verification**: ASI's `this.*` traces omit the ERC-20 and Ownable legs and imply self-calls. The only external interaction is the endpoint call. The two unrelated Tempo `_validateQuoteSwapPath(super._quote(...))` builder failures do not affect this path.

Function: `initializeV120()`

This permissionless `reinitializer(4)` function stores the fixed EIP-712 name `"Frax USD"` and version `"1.2.0"`, clears the hashed fields, and emits `Initialized(4)`, changing the accepted signature domain. Any caller may time the transition. Proxies at versions `0`–`2` can skip version-3 initialization and permanently disable `initialize(address)`.

Branches and code coverage

Intended branches

- Intended version-3 upgrade succeeds: an inactive proxy at version 3 advances to 4, stores the fixed domain fields, clears the initialization context, and emits `Initialized(4)`.
  - ☐ Test coverage

- Any caller can trigger the available transition: caller identity affects only submission timing and cannot alter the stored strings or acquire authority or value.
  - ☐ Test coverage
- Earlier initialization versions also satisfy the modifier: an inactive proxy at version 0, 1, or 2 can advance directly to 4, permanently preventing the version-3 `initialize(address)` wrapper.
  - ☐ Test coverage
- Concrete internal dispatch is deterministic: the `name()` and `version()` functions return `"Frax USD"` and `"1.2.0"`. The internal EIP-712 helpers omit `onlyInitializing`, but they are not ABI exposed, and `reinitializer(4)` encloses this call.
  - ☐ Test coverage

Negative behavior

- A proxy already initialized to version 4 or greater reverts before any EIP-712 write or successful `Initialized(4)` event.
  - ☐ Negative test
- An active initialization context causes the modifier to reject the nested attempt before the helper executes.
  - ☐ Negative test
- A direct implementation call fails because `_disableInitializers()` permanently disables initialization in implementation storage.
  - ☐ Negative test
- A call with nonzero `msg.value` is rejected by the nonpayable entry point without persistent effects.
  - ☐ Negative test
- Any internal failure or out-of-gas revert atomically rolls back the version, context, EIP-712 writes, and event.
  - ☐ Negative test

Function call analysis

- `this.__EIP712_init(this.name(), this.version()) -> this.__EIP712_init_unchained(name, version) -> this._getEIP712Storage()`
  - The generated `this.` qualifiers are incorrect: these are deterministic internal pure/storage calls, and the assembly only binds the fixed EIP-712 namespace slot without callbacks, value transfer, or untrusted return data.
- **Manual call-path verification**: Proxy fallback may delegatecall into `reinitializer(4)`, followed by internal `name()`, `version()`, EIP-712 storage updates, modifier cleanup, and `Initialized(4)`. Reverts propagate atomically, while builder failures at

`FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` concern separate Tempo paths and do not alter this call graph.

Function: `pause()`

The `pause()` function requires the immediate caller to be the stored owner, and then sets the shared namespaced pause flag through linked library code and emits `Paused()`. It accepts no parameters or value. Its behavior depends on existing ownership, pause-slot state, initialized proxy storage, and deployed library bytecode matching the reviewed source.

Branches and code coverage

Intended branches

- The first authorized pause changes the shared pause flag from false to true.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Pause_succeeds`)
- The successful pause emits `Paused()`.
  - ☐ Test coverage
- Paused-state enforcement for a non-owner token operation: after pausing, token hooks revert with `BeforeTokenTransferBlocked()` when the immediate caller is not the owner.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_BeforeTokenTransferBlocked_reverts`)
- Immediate-owner hook exemption: when the token-hook caller is the owner, all pause and freeze checks are skipped before calling the inherited hook.
  - ☐ Test coverage

Negative behavior

- A non-owner caller is rejected by `onlyOwner`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Pause_OnlyOwner`)
- Reversion before delegatecall and absence of an event are not independently asserted.
  - ☐ Negative test
- Repeating pause on an already paused token reverts with `IsPaused()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Pause_IsPaused_reverts`)
- Reversion before writes or event emission is not independently asserted.
  - ☐ Negative test
- Nonzero native value: because `pause()` is nonpayable, nonzero `msg.value` is rejected

before modifier or body execution.

- ☐ Negative test

- Linked-library failure: any delegatecall failure bubbles up, and transaction atomicity rolls back all writes and logs.
  - ☐ Negative test

Function call analysis

- `this._pause() -> PauseLib.pause()`
  - The generated `this._pause()` notation is inaccurate: internal virtual dispatch resolves `_pause()` to `PauseModule._pause()`, which delegatecalls the statically linked `PauseLib.pause()` without caller-controlled arguments.
- **Manual call-path verification**: The verified path is `FrxUSDOFTUpgradeable.pause()` → `PauseModule._pause()` → library `DELEGATECALL` to `PauseLib.pause()`. Delegatecall preserves context and `msg.sender`; the library performs no callbacks or further external calls, and failures bubble atomically. The concern about freezing `address(0)` is separate. Parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` affect unrelated Tempo `_quote` paths.

Function: `removeFreezer(address account)`

After `onlyOwner` validates `msg.sender`, the function removes `account` through `FreezeThawModule._removeFreezer` and linked `FreezeThawLib.removeFreezer`, emitting `RemoveFreezer(account)`. If `account` is not a freezer, the call reverts with `NotFreezer()`. Proxy routing, ownership initialization, and correct library linking are deployment assumptions.

Inputs

- `account`
  - **Control**: The caller selects `account`, but removal succeeds only when the stored owner calls.
  - **Constraints**: Any address, including zero, is accepted, but it must currently belong to the freezer set.
  - **Impact**: Successful removal may reorder `EnumerableSet` enumeration and emits the indexed address without thawing it.

Branches and code coverage

Intended branches

- With owner authorization and existing membership, removal persists and the freezer

list becomes empty.

- ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_RemoveFreezer_succeeds`)

- Success emits `RemoveFreezer(account)` and returns empty data.
  - ☐ Test coverage
- If `address(0)` is a freezer, it follows the same successful removal and event path.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` reverts at the nonpayable entry check before authorization or delegation.
  - ☐ Negative test
- An unauthorized caller is rejected.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_RemoveFreezer_OnlyOwner`)
- Reversion before `_removeFreezer` and nonuse of `tx.origin` are not independently instrumented.
  - ☐ Negative test
- Removing a nonmember reverts with `NotFreezer()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_RemoveFreezer_NotFreezer_reverts`)
- Postrevert storage and log state are not independently asserted.
  - ☐ Negative test
- Any linked-library failure propagates and atomically rolls back storage changes and emitted logs.
  - ☐ Negative test

Function call analysis

- `this._removeFreezer(account) -> FreezeThawLib.removeFreezer(account)`
  - The generated `this._removeFreezer(account)` edge is incorrect because dispatch is internal through `FreezeThawModule._removeFreezer`. Only the fixed linked-library edge is a compiler-generated `DELEGATECALL`, which preserves `msg.sender` and contract storage while performing set removal, conditional revert, and event emission without returns or further calls.
- **Manual call-path verification**: The verified path is `FrxUSDOFTUpgradeable.removeFreezer(account)` → `FreezeThawModule._removeFreezer(account)` →

`FreezeThawLib.removeFreezer(account)` → `EnumerableSet.AddressSet.remove(account)`. Matching storage locations ensure the contract's freezer set is mutated, while ASI builder errors at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern unrelated Tempo resolution.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This owner-only, nonpayable setter replaces the namespaced default through linked RateLimiterLib delegatecall in the caller/proxy context and emits `DefaultRateLimitConfigSet`. It returns nothing.

Inputs

- `_defaultConfig`
  - **Control**: Calldata supplies all seven fields, while the current owner and zero `msg.value` gate execution and no input selects the linked target.
  - **Constraints**: ABI decoding must succeed, and each enabled direction requires a nonzero matching limit and window.
  - **Impact**: The complete struct is stored, six fields are emitted with limits widened losslessly, and the default record's `overrideDefaultConfig` flag does not control EID selection.

Branches and code coverage

Intended branches

- With both directions disabled, validation permits zero limits or windows before storing the exact struct and emitting the event.
  - ☐ Test coverage
- With outbound enabled, both `outboundLimit` and `outboundWindow` must be nonzero.
  - ☐ Test coverage
- With inbound enabled, both `inboundLimit` and `inboundWindow` must be nonzero.
  - ☐ Test coverage
- EIDs without per-EID overrides use the replacement default, while enabled per-EID overrides remain effective.
  - ☐ Test coverage
- No EID is checkpointed, so later state calculation may apply the replacement limit and window to usage and elapsed time retained from the prior default.
  - ☐ Test coverage

- Success only writes rate-limiter storage and emits `DefaultRateLimitConfigSet` through library-context execution.
  - ☐ Test coverage

Negative behavior

- A caller other than the current owner fails `onlyOwner` before entering the setter body.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point.
  - ☐ Negative test
- Malformed, truncated, noncanonical, or out-of-range ABI input reverts during decoding.
  - ☐ Negative test
- Enabling outbound with a zero outbound limit or window reverts before assignment and emission.
  - ☐ Negative test
- Enabling inbound with a zero inbound limit or window reverts before assignment and emission.
  - ☐ Negative test
- Validation, out-of-gas, or another delegatecall failure reverts all storage changes and logs atomically.
  - ☐ Negative test

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`
  - The generated `this.` hop is incorrect; the source invokes inherited nonvirtual `_setDefaultRateLimitConfig` internally then the deployment-linked library by `DELEGATECALL`, with no caller-selected target, return-data dependency, callback, or further external interaction.
- **Manual call-path verification**: The resolved path is `setDefaultRateLimitConfig` → `RateLimiterModule._setDefaultRateLimitConfig` → linked `RateLimiterLib.setDefaultRateLimitConfig`, which validates, writes namespaced caller/proxy storage, and emits from the calling address. Unrelated send/payment builder failures do not alter this path.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This owner-only batch setter checkpoints each EID under its old effective configuration before storing validated replacements. It ignores global disable.

Inputs

- `_params`
  - **Control**: The owner selects the array length, EIDs, override flags, enabled directions, limits, and windows.
  - **Constraints**: Enabled directions require nonzero limits and windows, with no length, uniqueness, EID, ordering, or normalization checks.
  - **Impact**: Each element checkpoints and overwrites its EID sequentially, so duplicates observe earlier writes and any failure reverts the batch.
- `msg.sender` and current owner storage
  - **Control**: The immediate caller supplies `msg.sender`, while ownership comes from existing Ownable storage.
  - **Constraints**: The `onlyOwner` modifier requires `msg.sender` to equal the current owner.
  - **Impact**: Authorization succeeds only for the owner, and `msg.sender` remains unchanged through delegatecall.
- Prior per-EID configuration and default configuration
  - **Control**: Prior authorized transactions establish these values, except that earlier duplicate elements may update the per-EID entry.
  - **Constraints**: An override selects `configs[eid]`; otherwise `defaultConfig` applies, without consulting global disable.
  - **Impact**: The selected old configuration determines checkpoint decay before replacement storage.
- Prior per-EID rate-limit state
  - **Control**: Earlier consumption, checkpoints, administrative updates, or duplicate elements establish this state.
  - **Constraints**: The setter does not require existing usage to fit the old or new limits.
  - **Impact**: Prior usage and timestamp determine checkpointed values, and usage may remain above a new limit.
- `msg.value`
  - **Control**: The caller chooses the transaction value.
  - **Constraints**: The nonpayable entry point requires zero value.
  - **Impact**: No ETH is read, forwarded, or transferred.
- `block.timestamp`

    - **Control**: Consensus and block production determine the timestamp.
    - **Constraints**: It is narrowed to `uint32`, assuming migration before the year-2106 boundary.
    - **Impact**: It determines elapsed-time decay under the old effective configuration.
- Available transaction gas
    - **Control**: The submitter selects the gas limit within protocol constraints.
    - **Constraints**: No explicit batch-size or gas guard exists.
    - **Impact**: Out-of-gas reverts all preceding writes and logs.
- `tx.origin`
    - **Control**: The originating transaction determines it.
    - **Constraints**: This path never reads it.
    - **Impact**: It has no authorization, accounting, storage, event, or interaction effect.

Branches and code coverage

Intended branches

- An authorized empty batch executes no iterations, writes, or events.
    - ☐ Test coverage
- A valid element checkpoints under the old configuration, stores the replacement, and emits both events in that order.
    - ☐ Test coverage
- An existing override selects the per-EID configuration; otherwise, the default configuration applies.
    - ☐ Test coverage
- A zero or nonforward prior timestamp preserves usage and updates `lastUpdated`.
    - ☐ Test coverage
- Zero usage, limit, window, or elapsed time preserves that direction's usage while advancing the shared timestamp.
    - ☐ Test coverage
- Elapsed time at least equal to the old window reduces applicable usage to zero.
    - ☐ Test coverage
- Partial decay subtracts `oldLimit * elapsed / oldWindow`, capped at zero; disabled zero fields preserve usage, and shared-timestamp updates can discard fractional recovery.
    - ☐ Test coverage

- Duplicate EIDs execute sequentially, allowing earlier writes to affect later checkpoint configuration selection.
  - ☐ Test coverage
- Global disable does not stop validation, checkpointing, storage updates, or events.
  - ☐ Test coverage

Negative behavior

- A non-owner caller reverts before the helper or library path.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point.
  - ☐ Negative test
- Malformed dynamic-array or tuple ABI data fails decoding.
  - ☐ Negative test
- Enabled outbound limiting with a zero limit or window reverts before checkpointing that element.
  - ☐ Negative test
- Enabled inbound limiting with a zero limit or window reverts before checkpointing that element.
  - ☐ Negative test
- Any validation, arithmetic, delegatecall, storage, event, later-iteration, or gas failure reverts all batch effects.
  - ☐ Negative test
- Duplicate EIDs, EID zero, type-valid extremes, disabled zero fields, excessive existing usage, and poor owner policy are intentionally accepted.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
  - The owner-controlled values reach a fixed linked library via `DELEGATECALL`, which returns nothing, operates in caller storage/log context, propagates failure atomically, and makes no callbacks or further external calls.
- **Manual call-path verification**: The actual path is `FrxUSDOFTUpgradeable.setRateLimitConfigs` → internal `RateLimiterModule._setRateLimitConfigs` → linked `RateLimiterLib.setRateLimitConfigs` via `DELEGATECALL`. ASI's generated `this._setRateLimitConfigs` is not an ABI self-call.

Function: `setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)`

This owner-only setter uses a delegatecall to the linked RateLimiterLib to store the global switch in the target or proxy's namespaced storage and emit the corresponding event. Disabling rate limits bypasses consumption and reports unlimited availability. Reenabling them restores timestamp-decayed enforcement while preserving usage and shared per-EID bucket contention.

Inputs

- `_globalConfig`
  - **Control**: The caller supplies the struct's sole field, `bool isGloballyDisabled`, subject to `msg.sender == owner()`.
  - **Constraints**: Calldata must canonically encode the struct, `msg.value` must be zero, and both boolean values are accepted without further validation.
  - **Impact**: The library stores the boolean and emits `RateLimitGlobalConfigSet`, while preserving configurations, usage, and timestamps.

Branches and code coverage

Intended branches

- A call where `msg.sender == owner()` lets the decoded struct reach the internal setter.
  - ☐ Test coverage
- If `isGloballyDisabled` is `true`, the setter stores `true`, emits the event, bypasses consumption, and reports unlimited availability.
  - ☐ Test coverage
- If `isGloballyDisabled` is `false`, the setter stores `false`, emits the event, and restores configured timestamp-decayed enforcement.
  - ☐ Test coverage
- Successful `RateLimiterLib.setRateLimitGlobalConfig` execution commits the storage assignment and event.
  - ☐ Test coverage
- The `RateLimiterModuleTest` tests passed 6/6, but no exact per-branch mapping was supplied.
  - ☐ Test coverage

Negative behavior

- A call where `msg.sender != owner()` reverts before any storage change or event.

- ☐ Negative test
- Short, malformed, or noncanonical boolean calldata reverts during ABI decoding.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point.
  - ☐ Negative test
- Delegatecall revert or out-of-gas propagates and atomically rolls back the write and event.
  - ☐ Negative test
- Neither boolean value is rejected because both `true` and `false` are valid configurations.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The source first calls `_setRateLimitGlobalConfig` internally and then delegatecalls the fixed linked library. The delegatecall preserves `msg.sender`, the target's storage, and the target's address, and it exposes no callback or application-level return value.
- **Manual call-path verification**: The resolved path is `FrxUSDOFTUpgradeable.setRateLimitGlobalConfig` → internal `RateLimiterModule._setRateLimitGlobalConfig` → linked `RateLimiterLib.setRateLimitGlobalConfig`. The generated `this._setRateLimitGlobalConfig` notation implies an external self-call that does not occur.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

This owner-only, nonpayable setter delegates `_params` to the linked RateLimiterLib, replacing namespaced per-EID states without limit checks or decay checkpointing. It returns nothing.

Inputs

- `_params`
  - **Control**: After owner authorization, the caller controls the ordered array and each `eid`, `uint112` usage value, and `uint32 lastUpdated`, while the helper, selector, library target, storage namespace, and event signature remain fixed.

- **Constraints**: ABI decoding enforces declared widths. Empty, duplicate, unknown, and uncapped-length batches are allowed, while `lastUpdated` cannot be in the future and a zero timestamp requires both usages to be zero.
- **Impact**: Each valid element replaces its entire namespaced per-EID state and emits `RateLimitStateSet`. Duplicates each emit and the last wins. Stored usage and the shared timestamp govern later bidirectional capacity and decay.

Branches and code coverage

Intended branches

- An authorized empty batch performs no writes or events and succeeds.
  - ☐ Test coverage
- A zero-timestamp element with both usage values zero stores the complete zero state and emits.
  - ☐ Test coverage
- An element with `0 < lastUpdated <= block.timestamp` stores and emits any ABI-valid usage values without consulting configured limits.
  - ☐ Test coverage
- Multiple valid elements execute in order, with distinct EIDs updating separately and duplicate EIDs emitting each time before the last value prevails.
  - ☐ Test coverage
- The wrapper resolves through the inherited nonvirtual helper to the fixed linked library and cannot redirect that path through `_params`.
  - ☐ Test coverage

Negative behavior

- A non-owner reverts before the helper or library executes, producing no writes or events.
  - ☐ Negative test
- An element with `lastUpdated > block.timestamp` reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- A zero-timestamp element with either usage nonzero reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- Nonzero value is rejected as nonpayable, while malformed or out-of-width calldata fails ABI decoding.

- ☐ Negative test

- Any later validation failure, library failure, or out-of-gas condition propagates and rolls back all earlier batch writes and logs.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
  - The generated `this` prefix is incorrect because the helper call is internal. The fixed library edge is a delegatecall that preserves active storage, address, sender, and value, and the supplied library performs validation, namespaced writes, and events without further calls or callbacks.
- **Manual call-path verification**: The actual path is `FrxUSDOFTUpgradeable.setRateLimitStates` → internal nonvirtual `RateLimiterModule._setRateLimitStates` → delegatecall to fixed `RateLimiterLib.setRateLimitStates` → per-element validation, replacement, and emission. Failures propagate atomically, and ASI's trace omits delegatecall context, duplicate behavior, and rollback.

Function: `thaw(address account)`

The owner-only `thaw` function forwards `account` unchanged through the inherited `_thaw` function to linked `FreezeThawLib.thaw`. It idempotently removes the address when present and always emits unindexed `AccountThawed(account)`.

Inputs

- `account`
  - **Control**: The owner controls the address forwarded unchanged as the frozen-set key and event argument.
  - **Constraints**: Zero and already thawed addresses are allowed, while `onlyOwner` constrains `msg.sender` rather than `account`.
  - **Impact**: Removal unfreezes the address when present, and thawing zero allows minting and burning that freezing zero would block.

Branches and code coverage

Intended branches

- The owner removes a currently frozen tested account from namespaced frozen

storage.

- ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Thaw_succeeds`)

- Success emits `AccountThawed(account)`.
  - ☐ Test coverage
- Owner thaws an account that is not frozen: the ignored removal result makes the operation succeed without mutation while still emitting `AccountThawed(account)`.
  - ☐ Test coverage
- Owner thaws `address(0)`: zero follows the same present-or-absent removal behavior and emits `AccountThawed(address(0))`.
  - ☐ Test coverage
- The supplied nonzero account is removed from frozen membership.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Thaw_succeeds`)
- Exact event argument forwarding remains unasserted.
  - ☐ Test coverage

Negative behavior

- A non-owner caller is rejected; a freezer role alone does not authorize thaw.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Thaw_OnlyOwner`)
- Postrevert state and event absence are not independently asserted.
  - ☐ Negative test
- Caller attaches native value: Solidity's nonpayable check rejects nonzero `msg.value` before function execution.
  - ☐ Negative test
- Linked-library execution reverts: the revert bubbles and atomically rolls back any set mutation and event emission.
  - ☐ Negative test
- The `tx.origin` value differs from the owner: authorization depends only on `msg.sender`, so an owner origin cannot authorize a non-owner intermediary.
  - ☐ Negative test

Function call analysis

- `this._thaw(account) -> FreezeThawLib.thaw(account)`
  - Resolved edge: The generated `this._thaw(account)` is source-inaccurate because the source uses internal virtual dispatch to `FreezeThawModule._thaw`, followed by a linked-library `DELEGATECALL` whose

expected bytecode is a deployment trust assumption and which exposes no callback or reentrancy point.

- **Manual call-path verification**: The path is `FrxUSDOFTUpgradeable.thaw(account)` → `FreezeThawModule._thaw(account)` → `FreezeThawLib.thaw(account)` → `$.frozen.remove(account)` → `AccountThawed(account)`. Parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` affect sibling Tempo paths only.

Function: `thawMany(address[] accounts)`

This owner-only, nonpayable bulk thaw delegates to the linked FreezeThawLib, removing supplied addresses from namespaced frozen storage and emitting `AccountThawed` per element. Removal is idempotent, but events are unconditional.

Inputs

- `accounts`
  - **Control**: The owner controls array length, order, duplicates, and values including `address(0)`, while neither `tx.origin` nor callee-supplied values affect processing.
  - **Constraints**: Calldata must decode as `address[]`, `msg.value` must be zero, and `msg.sender` must satisfy `onlyOwner`. Batch length is bounded only by gas.
  - **Impact**: Present entries are removed, absent or duplicate entries still emit, and thawing `address(0)` reenables otherwise blocked non-owner mint and burn paths, including OFT credit and debit.

Branches and code coverage

Intended branches

- Authorization-success branch: an owner call forwards the complete array through `_thawMany` to `FreezeThawLib.thawMany`.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_ThawMany_succeeds`)
- Empty-array branch: an empty array performs no iterations, mutations, or events and succeeds.
  - ☐ Test coverage
- Each currently frozen tested account is removed from membership.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_ThawMany_succeeds`)
- Each removal emits `AccountThawed(account)`.

- ☐ Test coverage

- Absent-account branch: an absent account changes no membership but still emits `AccountThawed(account)`.
  - ☐ Test coverage
- Duplicate-account branch: the first occurrence may remove membership, while every occurrence emits an event in input order.
  - ☐ Test coverage
- Zero-address branch: an `address(0)` entry is accepted, removed when present, and emits an event.
  - ☐ Test coverage
- Completion branch: successful iteration returns without a declared return value.
  - ☐ Test coverage

Negative behavior

- A non-owner caller is rejected.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_ThawMany_OnlyOwner`)
- Reversion before dispatch, mutation, or event emission is not instrumented.
  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable entry point.
  - ☐ Negative test
- Malformed `address[]` calldata reverts before the function body executes.
  - ☐ Negative test
- Delegatecall failure, panic, or out-of-gas reverts atomically, rolling back all prior removals and logs.
  - ☐ Negative test
- The supplied verification summary contains no exact `thawMany` test or per-branch mapping.
  - ☐ Negative test

Function call analysis

- `this._thawMany(accounts) -> FreezeThawLib.thawMany(accounts)`
  - The source dispatches internally from `_thawMany(accounts)` to `FreezeThawModule._thawMany` then makes a fixed-target library `DELEGATECALL`. The delegatecall preserves token or proxy storage, `address(this)`, and `msg.sender`. The library performs only set removal, looping, and event emission. Failures revert all effects, and there are no

callbacks or reentrancy-capable calls.

- **Manual call-path verification**: Sources confirm `FrxUSDOFTUpgradeable.thawMany` → `FreezeThawModule._thawMany` → `FreezeThawLib.thawMany` → per-element `thaw`. ASI renders the initial call as `this._thawMany`. Sibling builder errors at `contracts/FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `contracts/FraxOFTUpgradeableTempo.sol@48:2180` do not affect this path.

Function: `unpause()`

The owner-only, nonpayable `unpause()` function internally resolves to `PauseModule._unpause()`, which delegatecalls link-time-bound `PauseLib.unpause()` to clear a true namespaced pause flag and emit `Unpaused()`. Unpausing restores non-owner token operations while freezes remain enforced. Owner-originated hook operations already bypass both checks.

Inputs

Parameter taint

- There are no explicit calldata parameters or return values.
- The `msg.sender` value is caller-controlled and used only by the inherited `onlyOwner` modifier. Delegatecall preserves it, but the reviewed library does not read it.
- The attached `msg.value` must be zero because the function is nonpayable, and no ETH is forwarded or retained.
- The `tx.origin` value is unread and does not affect authorization, storage, events, or control flow.
- The configured owner is an implicit storage input used only for authorization.
- The namespaced `isPaused` flag determines whether execution clears the flag or reverts with `PauseModule.NotPaused()`.
- The reviewed PauseLib address is fixed in implementation bytecode at link time and cannot be selected by callers.

Branches and code coverage

Intended branches

- An authorized caller clears the pause flag from true to false.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Unpause_succeeds`)
- Success emits `Unpaused()` and returns no data.
  - ☐ Test coverage
- The tested concrete target reaches the inherited unpause path and changes paused state to false.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Unpause_succeeds`)

- Absence of overrides in every supplied descendant is a source property, not separately tested.

  - ☐ Test coverage

Negative behavior

- An unauthorized caller is rejected.

  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Unpause_OnlyOwner`)
- Unchanged state and absence of persistent events are not independently asserted.

  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable entry check before authorization or state access.

  - ☐ Negative test
- An authorized call while already unpaused reverts with `NotPaused()`.

  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Unpause_NotPaused_reverts`)
- Reversion before writes or event emission is not independently asserted.

  - ☐ Negative test
- Any other delegatecall failure propagates and rolls back all state and events.

  - ☐ Negative test

Function call analysis

- `this._unpause() -> PauseLib.unpause()`

  - The generated `this._unpause()` trace is incorrect. The source uses internal `_unpause()`, which resolves to `PauseModule._unpause()` and then a fixed-target `DELEGATECALL` to argumentless `PauseLib.unpause()`. The library accesses only the pause slot, makes no further calls, returns no value, and propagates failures.
- **Manual call-path verification**: The path is nonpayable check → `onlyOwner` → `FrxUSDOFTUpgradeable.unpause()` → `PauseModule._unpause()` → linked `PauseLib.unpause()`. No ETH, token, LayerZero, callback, or reentrancy edge exists, and ASI builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` concern unrelated Tempo send/fee paths.

5.19. Module: FrxUSDPolicyAdminTempo.sol

Function: `addFreezer(address account)`

This function grants the freezer role to `account`, authorizing it to call `freeze` and `freezeMany`.

Inputs

- `account`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Reverts with `AlreadyFreezer` if already a freezer.
  - **Impact**: The `account` address can freeze, but not thaw, any address.

Branches and code coverage

Intended branches

- The `account` address is added to the `freezers` mapping, and `AddFreezer` is emitted.
  - ☑ Test coverage (`test_AddFreezer_Succeeds`, `test_AddFreezer_EmitsEvent`)

Negative behavior

- The function reverts when the caller is not the owner.
  - ☑ Negative test (`FrxUSDPolicyAdminTempoTest.test_AddFreezer_OnlyOwner`)
- The function reverts when `account` is already a freezer.
  - ☑ Negative test (`test_AddFreezer_AlreadyFreezer_Reverts`)
- The function does not revert when `account == address(0)`.
  - ☐ Negative test

Function: `freeze(address account)`

This function adds `account` to the TIP-403 blacklist.

Inputs

- `account`
  - **Control**: Fully controlled by the caller (any freezer or owner).
  - **Constraints**: No zero-address check.
  - **Impact**: The `account` address is blacklisted in the TIP-403 policy.

Branches and code coverage

Intended branches

- The `account` address is added to the TIP-403 blacklist.

- ☑ Test coverage (`test_Freeze_AsOwner_Succeeds`, `test_Freeze_AsFreezer_Succeeds`)

- The `AccountFrozen` event is emitted.
  - ☑ Test coverage (`test_Freeze_EmitsEvent`)

Negative behavior

- The function reverts when the caller is neither a freezer nor the owner.
  - ☑ Negative test (`test_Freeze_NotFreezerOrOwner_Reverts`)

Function: `freezeMany(address[] accounts)`

This function batch-freezes multiple accounts in a single loop.

Inputs

- `accounts`
  - **Control**: Fully controlled by the caller (any freezer or owner).
  - **Constraints**: None.
  - **Impact**: Each element is blacklisted.

Branches and code coverage

Intended branches

- All `accounts` are blacklisted.
  - ☑ Test coverage (`test_FreezeMany_AsOwner_Succeeds`, `test_FreezeMany_AsFreezer_Succeeds`)
- The `AccountFrozen` event is emitted for each element.
  - ☑ Test coverage (`test_FreezeMany_EmitsEvents`)

Negative behavior

- The function reverts when the caller is neither a freezer nor the owner.
  - ☑ Negative test (`test_FreezeMany_NotFreezerOrOwner_Reverts`)
- Duplicate addresses in `accounts` cause repeated `AccountFrozen` events with no state change after the first.
  - ☐ Negative test

Function: `initialize(address _owner)`

This function initializes the proxy, sets the owner, and creates a new `BLACKLIST` policy in the TIP-403 Registry with `address(this)` as admin.

Inputs

- `_owner`
  - **Control**: Fully controlled by the deployer.
  - **Constraints**: No zero-address check.
  - **Impact**: Becomes the sole account with owner-level authority.

Branches and code coverage

Intended branches

- The owner is set correctly.
  - ☑ Test coverage (`test_Initialize_SetsOwner`)
- A BLACKLIST policy is created, and `policyId > 0` is stored.
  - ☑ Test coverage (`test_Initialize_CreatesPolicyId`, `test_GetPolicyData_ReturnsBlacklist`)
- The `PolicyIdSet` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function reverts when already initialized.
  - ☑ Negative test (`test_Initialize_CannotReinitialize`)

Function: `initializeWithPolicy(address _owner, uint64 _policyId)`

This function initializes the proxy using a preexisting TIP-403 policy, storing `_policyId` directly without on-chain validation.

Inputs

- `_owner`
  - **Control**: Fully controlled by the deployer.
  - **Constraints**: No zero-address check.
  - **Impact**: Becomes the sole account with owner-level authority.

- `_policyId`
  - **Control**: Fully controlled by the deployer.
  - **Constraints**: None.
  - **Impact**: All `freeze` / `thaw` calls route to this ID.

Branches and code coverage

Intended branches

- The owner is set, and `policyId` is stored.
  - ☑ Test coverage (`test_InitializeWithPolicy_SetsOwner`)
- The `PolicyIdSet` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function does not revert when `_policyId == 0`, which silently disables all freeze/thaw operations.
  - ☐ Negative test
- The function does not verify that `address(this)` is the admin of `_policyId`.
  - ☐ Negative test
- The function does not verify that the policy type is `BLACKLIST`.
  - ☐ Negative test

Function: `removeFreezer(address account)`

This function revokes the freezer role from `account`.

Inputs

- `account`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Reverts with `NotFreezer` if `account` does not hold the role.
  - **Impact**: The `account` address can no longer call `freeze` or `freezeMany`.

Branches and code coverage

Intended branches

- The `account` address is removed from the `freezers` mapping, and `RemoveFreezer` is emitted.
  - ☑ Test coverage (`test_RemoveFreezer_Succeeds`, `test_RemoveFreezer_EmitsEvent`)

Negative behavior

- The function reverts when the caller is not the owner.
  - ☑ Negative test (`FrxUSDPolicyAdminTempoTest.test_RemoveFreezer_OnlyOwner`)
- The function reverts when `account` is not a freezer.
  - ☑ Negative test (`test_RemoveFreezer_NotFreezer_Reverts`)

Function: `setPolicyAdmin(address newAdmin)`

This function transfers TIP-403 policy admin to `newAdmin` in a single step.

Inputs

- `newAdmin`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No zero-address check.
  - **Impact**: The old admin loses all admin authority over the TIP-403 policy, which passes to `newAdmin`.

Branches and code coverage

Intended branches

- The policy admin is transferred to `newAdmin`.
  - ☑ Test coverage (`test_SetPolicyAdmin_Succeeds`)

Negative behavior

- The function reverts when the caller is not the owner.
  - ☑ Negative test (`test_SetPolicyAdmin_OnlyOwner`)

Function: `thaw(address account)`

This function removes `account` from the TIP-403 blacklist.

Inputs

- `account`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: The `account` address is removed from the blacklist.

Branches and code coverage

Intended branches

- The `account` address is removed from the TIP-403 blacklist.
  - ☑ Test coverage (`test_Thaw_Succeeds`)
- The `AccountThawed` event is emitted.
  - ☑ Test coverage (`test_Thaw_EmitsEvent`)

Negative behavior

- The function reverts when the caller is not the owner (freezers cannot thaw).
  - ☑ Negative test (`FrxUSDPolicyAdminTempoTest.test_Thaw_OnlyOwner`)

Function: `thawMany(address[] accounts)`

This function batch-thaws multiple accounts. It is owner-only and uses the same loop pattern as `freezeMany`.

Inputs

- `accounts`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: All `accounts` are removed from the blacklist.

Branches and code coverage

Intended branches

- All `accounts` are removed from the blacklist.
  - ☑ Test coverage (`test_ThawMany_Succeeds`)
- The `AccountThawed` event is emitted for each element.

- ☑ Test coverage (`test_ThawMany_EmitsEvents`)

Negative behavior

- The function reverts when the caller is not the owner.
  - ☑ Negative test (`FrxUSDPolicyAdminTempoTest.test_ThawMany_OnlyOwner`)
- Duplicate addresses emit repeated `AccountThawed` events with no additional state change.
  - ☐ Negative test

5.20. Module: HopV201Tempo.sol

Function: `constructor()`

The `HopV201Tempo` constructor disables initialization once. The concrete `RemoteHopV201Tempo(_endpoint)` constructor disables it twice and immutably stores the nonzero token reported by the trusted endpoint. Deployment and endpoint selection are trusted and unauthenticated.

Inputs

- Direct construction has no parameters. Concrete deployment forwards the deployer-controlled `_endpoint` unchanged to TempoGasTokenBase; its zero-value static `nativeToken()` call must return a decodable, nonzero address to store in immutable `nativeToken`.
- The constructors do not use `msg.sender` or `tx.origin`, are nonpayable, perform no arithmetic, and emit no source-visible events. Separately, the wrapper input `_hop` must be nonzero and have deployed code before assignment to immutable `HOP`.

Branches and code coverage

Intended branches

- Direct HopV201Tempo creation: the internal `_disableInitializers()` helper executes once, and any revert atomically aborts creation.
  - ☐ Test coverage
- Concrete RemoteHopV201Tempo construction succeeds against a nonzero endpoint-native-token result and yields an initialized proxy implementation.
  - ☑ Test coverage (`RemoteHopV201TempoForkTest.testFork_RecoverERC20`, `Remote-HopV201TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectToFraxt`
- The test does not directly assert both initializer-lock calls.

- ☐ Test coverage

- Nonzero endpoint-native-token result: a decodable nonzero `nativeToken()` result is stored immutably and permits construction to continue.
  - ☑ Test coverage (`RemoteHopV201TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct`)
- Valid wrapper hop: nonzero `_hop` with code is interface-cast and stored unchanged in immutable `HOP`.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_Constructor_RejectsZeroOrCodelessHop`)
- No source-level conditional in either initializer-locking body: both locking constructor bodies contain only `_disableInitializers()`, while the zero-token branch resides in TempoGasTokenBase.
  - ☐ Test coverage

Negative behavior

- Nonzero creation value: nonpayable construction with native value fails and rolls back all creation effects.
  - ☐ Negative test
- Endpoint call failure or malformed return data: endpoint revert, gas exhaustion, or undecodable address data aborts concrete deployment.
  - ☐ Negative test
- Zero native-token result: a zero endpoint result reverts with `NativeTokenUnavailable()` before the derived constructor body.
  - ☐ Negative test
- Initializer helper failure: revert from either `_disableInitializers()` invocation propagates and atomically aborts deployment.
  - ☐ Negative test
- A zero or currently codeless wrapper target reverts with `InvalidHop(_hop)`.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_Constructor_RejectsZeroOrCodelessHop`)
- Absence of an external interface call is not independently instrumented.
  - ☐ Negative test

Function call analysis

- `_disableInitializers()`
  - This internal, argumentless, non-redirectable helper runs once during direct construction and twice during concrete construction. It transfers nothing

and propagates any revert.

- `TempoGasTokenBase(_endpoint)`
  - This base-constructor path forwards `_endpoint` unchanged and stores the token reported by the endpoint only if deployment completes. On failure, the derived constructor body does not execute.
- `IEndpointV2Alt(_lzEndpoint).nativeToken()`
  - This zero-value staticcall targets the deployer-selected endpoint and decodes the returned address. The endpoint can revert, exhaust gas, or determine the returned data; a zero address is rejected. It cannot execute the unfinished contract's runtime code.
- `_hop.code.length` and `HOP = IRemoteHopTempo(_hop)`
  - This independent wrapper path checks that `_hop` is nonzero and currently has code and then stores it without invoking or validating the declared interface.
- **Manual call-path verification**: Generated FCA omitted a sink header. The source shows these paths: direct disable; concrete disable → endpoint query/check → disable; and independent wrapper validation → `HOP` assignment.

Findings

- Constructor safety depends on trusted deployment, endpoint selection, and wrapper configuration.
- Fee selection and refunds, zero-recipient handling, quote sizing, replay identity, and queued-settlement custody affect runtime only and add no constructor branch or constructor-time asset movement.

Function: `pauseOff()`

The `pauseOff` function lets `DEFAULT_ADMIN_ROLE` holders idempotently set `paused = false` in the fixed ERC-7201 `HopV2Storage` slot. It is nonpayable, emits nothing, returns no data, and makes no external calls.

Branches and code coverage

Intended branches

- Authorized execution: with zero `msg.value`, a caller holding `DEFAULT_ADMIN_ROLE` writes `paused = false`.
  - ☐ Test coverage
- Prior state is paused: writing `false` changes the flag from `true` and reopens downstream pause gates without bypassing other checks.

- ☐ Test coverage

- Prior state is already unpaused: writing `false` again succeeds without changing state or emitting an event.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller lacking `DEFAULT_ADMIN_ROLE`, including one holding only `PAUSER_ROLE`, reverts before slot lookup or storage modification.
  - ☐ Negative test
- Native value supplied: nonzero `msg.value` reverts before authorization or storage modification, so no ETH is retained.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
  - The generated `this.` notation is inaccurate: `_getHopV2Storage()` is an internal pure helper whose Yul binds slot `0x6f2b5e4a4e4e1ee6e84aeabd150e6bcb39c4b05494d47809c3cd3d998f859100` without calls, callbacks, reentrancy, or a source-level revert path.
- **Manual call-path verification**: Source review confirms that nonpayable and role checks precede the fixed-slot write; unpausing permits queued-compose retry but does not restore recovered principal, clear replay markers, reapprove OFTs, or restore remote mappings, so separate reconciliation remains required.

Function: `pauseOn()`

Authorized callers set `HopV2Storage.paused` to `true`; repeated calls succeed without emitting events or returning data. While the hop is paused, the `HopV201Tempo.sendOFT` and `_validateComposeMessage` functions, the `RemoteHopV201Tempo.sendOFT` and `lzCompose` functions, and TempoFeeInclusiveWrapper sends reject with `HopPaused`. Pausing after credit but before queued compose delays settlement until an administrator calls `pauseOff`; it does not transfer or recover principal.

Branches and code coverage

Intended branches

- A `DEFAULT_ADMIN_ROLE` holder sets `paused = true`, after which the wrapper's quote and send gates fail closed.
  - ☑ Test coverage

(TempoFeeInclusiveWrapperForkTest.test_HopPaused_QuoteAndSendFailClosed

- If the caller lacks `DEFAULT_ADMIN_ROLE` but has `PAUSER_ROLE`, `paused` is set to `true`.
  - ☐ Test coverage
- If an authorized caller invokes `pauseOn` while already paused, the call succeeds and writes `true` again.
  - ☐ Test coverage

Negative behavior

- If `msg.sender` has neither role, `onlyAuthorized` reverts with `NotAuthorized()` before the helper or state write.
  - ☐ Negative test
- Because `pauseOn` is nonpayable, nonzero `msg.value` reverts before authorization and leaves state and value unchanged.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
  - ASI misrenders the inherited internal pure `_getHopV2Storage()` helper as an external self-call; it only assigns fixed slot `0x6f2b5e4a4e4e1ee6e84aeabd150e6bcb39c4b05494d47809c3cd3d998f859100`.
- **Manual call-path verification**: The actual path is nonpayable `pauseOn()` → `onlyAuthorized` → one or two inherited internal `hasRole` reads → internal pure `_getHopV2Storage()` → `paused = true`. There are no external calls or reentrancy, the pointer is uncontrollable, and `tx.origin` is unused; any revert atomically rolls back the write.

Function: `recoverERC20(address _tokenAddress, uint256 _tokenAmount)`

Administrator-only recovery transfers a caller-selected ERC-20 amount to the immediate role-bearing caller, then emits `RecoveredERC20`. It does not update Hop accounting, reserve queued-compose liabilities, or verify actual receipt. This creates a trusted-custody risk, because an administrator may withdraw principal needed for pending compose settlement.

Inputs

- `_tokenAddress`
  - **Control**: The authorized immediate caller selects the unvalidated token target, while `tx.origin` is ignored.

    - **Constraints**: SafeERC20 requires a successful call with acceptable optional return data but performs no Hop approval or liability check.
    - **Impact**: The target receives zero-value calldata for `transfer(msg.sender, _tokenAmount)` and may implement hostile or nonstandard behavior.
- `_tokenAmount`
    - **Control**: The authorized caller supplies the amount unchanged.
    - **Constraints**: No bounds, balance, allowance, or pending-liability checks apply, and zero is permitted locally.
    - **Impact**: The requested amount is transferred and emitted without proving the recipient's actual balance increase.
- The caller (`msg.sender`) must hold `DEFAULT_ADMIN_ROLE` and becomes the recipient; a role-bearing forwarding contract receives the tokens itself.
- The attached native value (`msg.value`) must be zero because the function is nonpayable.

Branches and code coverage

Intended branches

- An authorized caller transfers the requested ERC-20 amount from the hop to itself.
    - ☑ Test coverage (`HopV201TempoIntegrationTest.test_Integration_RecoverStuckTokens`, `RemoteHopV201TempoForkTest.testFork_RecoverERC20`)
- Emission and ordering of `RecoveredERC20` remain untested.
    - ☐ Test coverage
- Zero-amount boundary: a zero amount still invokes the selected token and emits zero if accepted.
    - ☐ Test coverage
- Optional-return success boundary: SafeERC20 accepts boolean `true` or empty return data from a successful call.
    - ☐ Test coverage
- Pending-compose principal recovery: no condition prevents an administrator from recovering principal credited before separate compose settlement.
    - ☐ Test coverage — the supplied threat-model verification modeled pre-compose credit and then exercised the recovery and compose paths in scope.
- Nonpayable boundary: Solidity rejects nonzero `msg.value` before role checking or body execution.
    - ☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE` reverts before recovering tokens.
  - ☑ Negative test (`RemoteHopV201TempoForkTest.testFork_RecoverERC20_NotAuthorized`)
- Token-level revert or rejected result: reverts, boolean `false`, malformed data, or failed low-level calls revert atomically without the Hop event.
  - ☐ Negative test
- Insufficient or nontransferable balance: a token rejection caused by insufficient or restricted balance reverts the transaction.
  - ☐ Negative test
- Zero, codeless, or nonconforming target: SafeERC20 rejects ordinary zero or codeless targets and unacceptable contract responses.
  - ☐ Negative test
- Pending-liability withdrawal: trusted administrators may underfund queued compose settlement until the balance is restored or claims are otherwise reconciled.
  - ☐ Negative test — the tested sequence recovers precompose credited principal and then executes the compose path

Function call analysis

- `SafeERC20.safeTransfer(IERC20(_tokenAddress), msg.sender, _tokenAmount)`
  - The wrapper makes one zero-value `transfer(msg.sender, _tokenAmount)` call; hostile tokens may lie, charge fees, rebase, call back, or reenter, while rejected results revert all state and logs.
- **Manual call-path verification**: SafeERC20 uses one internal low-level token call, which the generated trace compresses. There is no local accounting or reentrancy guard. A nested recovery rechecks the immediate caller's role.

Function: `sendOFT(address _oft, uint32 _dstEid, byte[32] _recipient, uint256 _amountLD, uint128 _dstGas, bytes _data)`

Any caller may route an approved OFT after pause and dust checks. Success emits `SendOFT` with the first-cleaned amount. The exported `byte[32]` type differs from the source `bytes32` type.

Inputs

- `_oft`
  - **Control**: The caller selects an administrator-approved OFT.

- **Constraints**: Unapproved OFTs, zero rates, or invalid dependency responses revert.
- **Impact**: The OFT controls conversion, token, quote, send, allowance, and callbacks.

- `_dstEid`
  - **Control**: The caller selects the destination EID.
  - **Constraints**: The configured local EID selects local delivery; others select remote delivery.
  - **Impact**: It controls routing, fee quoting, and the emitted destination.
- `_recipient`
  - **Control**: The caller selects arbitrary recipient bytes.
  - **Constraints**: No nonzero or destination-format validation occurs.
  - **Impact**: The low 160 bits select the local recipient and composer.
- `_amountLD`
  - **Control**: The caller selects the amount, subject to balance and allowance.
  - **Constraints**: Dust removal uses `decimalConversionRate()`; a zero rate reverts.
  - **Impact**: The first-cleaned amount controls transfers, approvals, composition, and emission.
- `_dstGas`
  - **Control**: The caller selects any `uint128` gas value.
  - **Constraints**: Nonempty data applies a `400_000` second-leg quoting floor.
  - **Impact**: It affects remote executor quotes but is unused locally.
- `_data`
  - **Control**: The caller supplies arbitrary bytes.
  - **Constraints**: Empty data skips local composition and compose-gas options.
  - **Impact**: Nonempty data reaches the composer and affects remote pricing.

Branches and code coverage

Intended branches

- Paused execution reverts with `HopPaused()`; otherwise, execution continues.
  - ☐ Test coverage
- Unapproved OFTs revert with `InvalidOFT()`; approved OFTs proceed.
  - ☐ Test coverage
- Dust removal floors amounts; a zero conversion rate reverts.
  - ☐ Test coverage

- Zero cleaned amounts skip pulling; positive amounts are pulled.
  - ☐ Test coverage
- The local EID invokes `_sendLocal`; other EIDs invoke `_sendToDestination`.
  - ☐ Test coverage
- Local positive amounts transfer; zero amounts skip payment.
  - ☐ Test coverage
- Nonempty local data invokes `hopCompose`; empty data skips it.
  - ☐ Test coverage
- The base remote hook returns a zeroed `SendParam` after second cleaning.
  - ☐ Test coverage
- This entry point always uses `quoteSend`; untrusted-message pricing is unreachable.
  - ☐ Test coverage
- Positive remote amounts invoke `forceApprove`; zero amounts skip approval.
  - ☐ Test coverage
- The `IOFT.send` call forwards native fees, and endpoint excess returns to this contract.
  - ☐ Test coverage
- The `quoteHop` call is skipped when either relevant EID is `FRAXTAL_EID`.
  - ☐ Test coverage
- Insufficient value reverts; equality succeeds; excess value is refunded.
  - ☐ Test coverage
- Success emits `SendOFT` with the first-cleaned amount and returns nothing.
  - ☐ Test coverage

Negative behavior

- Paused calls revert before OFT or token interactions.
  - ☐ Negative test
- Unapproved OFTs revert before conversion or token movement.
  - ☐ Negative test
- Invalid conversion calls or zero rates revert.
  - ☐ Negative test
- Failed or malformed principal transfers are rejected by SafeERC20.
  - ☐ Negative test
- Local transfer or composer failure reverts the initial pull.
  - ☐ Negative test

- Zero-equivalent local recipients remain allowed because no nonzero-recipient validation occurs.
  - ☐ Negative test
- Remote quote, approval, send, DVN, executor, or treasury failures revert.
  - ☐ Negative test
- Insufficient call `msg.value` reverts despite any existing contract balance.
  - ☐ Negative test
- Rejected excess-value refunds revert with `RefundFailed()`.
  - ☐ Negative test
- Checked fee, multiplier, length, or addition overflow reverts.
  - ☐ Negative test
- Token, OFT, composer, and refund calls may reenter before reconciliation.
  - ☐ Negative test
- Default-zero base parameters continue until downstream rejection.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
- `this.removeDust(_oft, _amountLD) -> IOFT2(oft).decimalConversionRate()`
- `SafeERC20.safeTransferFrom(IERC20(IOFT(_oft).token()), msg.sender, address(this), _amountLD)`
- `this._sendLocal(...) -> SafeERC20.safeTransfer(...) -> IHopComposer(recipient).hopCompose(...)`
- `this._sendToDestination(...) -> IOFT(_oft).quoteSend(sendParam, False)`
- `this._sendToDestination(...) -> SafeERC20.forceApprove(...) -> IOFT(_oft).send{value: fee.nativeFee}(...)`
- `this._sendToDestination(...) -> this.quoteHop(...) -> ILayerZeroDVN($.DVN).getFee(...)`
- `this._sendToDestination(...) -> this.quoteHop(...) -> IExecutor($.EXECUTOR).getFee(...)`
- `this._sendToDestination(...) -> this.quoteHop(...) -> ILayerZeroTreasury($.TREASURY).getFee(...)`
- **External edges**: Approved dependencies control tokens, rates, fees, callbacks, failures, and reentrancy; excess value returned by the endpoint goes to the hop, while executor pricing uses `360 + data.length` although execution encodes `40 + data.length`.
- **Manual call-path verification**: Storage Yul only selects the ERC-7201 slot. The base hook is empty, descendant traces are excluded, and `_handleMsgValue` checks the call's `msg.value` and then refunds excess. Alternate fee-token spend caps do not apply here,

while replay identity and queued-settlement custody arise only in inbound `lzCompose`.

Function: `sendOFT(address _oft, uint32 _dstEid, byte[32] _recipient, uint256 _amountLD)`

This overload dispatches to the six-argument override with zero gas and empty data, preserving the caller and value.

Inputs

- `_oft`
  - **Control**: Calldata selects `_oft`, subject to administrator approval.
  - **Constraints**: The OFT must provide valid token, rate, quote, and send behavior.
  - **Impact**: It determines principal, fees, allowances, delivery, and the emitted OFT address.
- `_dstEid`
  - **Control**: Calldata supplies the destination EID.
  - **Constraints**: Local, Fraxtal, and other remote EIDs select distinct routes.
  - **Impact**: It controls routing, fees, composition, and the emitted destination.
- `_recipient`
  - **Control**: Calldata controls the recipient without nonzero validation.
  - **Constraints**: Local delivery uses the low 160 bits; remote delivery uses full `bytes32`.
  - **Impact**: It selects delivery and permits a recipient whose low 160 bits decode to the zero address.
- `_amountLD`
  - **Control**: Calldata controls the amount, subject to balance and allowance.
  - **Constraints**: The amount is rounded down once; a zero rate reverts.
  - **Impact**: The cleaned amount controls transfers, allowances, sending, and event data.

Branches and code coverage

Intended branches

- Nonzero `msg.value` reverts immediately; zero proceeds.
  - ☐ Test coverage
- A paused hop reverts; an unpaused hop proceeds.

☐ Test coverage

- An unapproved OFT reverts; an approved OFT proceeds.

☐ Test coverage

- A zero conversion rate reverts; otherwise, dust is removed once.

☐ Test coverage

- A zero cleaned amount skips `transferFrom`; a positive amount invokes it.

☐ Test coverage

- The local EID selects `_sendLocal`; other EIDs select remote routing.

☐ Test coverage

- A local zero amount skips the transfer; positive amounts use the recipient's low 160 bits.

☐ Test coverage

- Local `hopCompose` is unreachable because this overload fixes empty data.

☐ Test coverage

- Fraxtal receives directly; other remote EIDs route through Fraxtal with composition.

☐ Test coverage

- Multihop routing raises zero gas to `400_000` and uses `1_000_000` compose gas.

☐ Test coverage

- Second-leg fees are zero at Fraxtal endpoints; otherwise, `quoteHop` applies.

☐ Test coverage

- Zero multipliers mean 1x; others scale by `multiplier / 10_000`.

☐ Test coverage

- Empty executor options use fallback options; gas appending remains unreachable.

☐ Test coverage

- Zero fees resolve a token; positive fees require collection.

☐ Test coverage

- Whitelisted tokens are collected directly; others use bounded exact-output swaps.

☐ Test coverage

- Failed quotes are skipped; no viable swap route reverts.

☐ Test coverage

- Excess swap input is refunded to the original caller.

☐ Test coverage

- Matching fee and principal tokens combine approval amounts so the OFT can spend both.

☐ Test coverage

- The `IOFT.send` call receives zero native value, and its returned data is ignored.
  - ☐ Test coverage
- Successful paths emit `SendOFT` with the cleaned amount and return nothing.
  - ☐ Test coverage

Negative behavior

- Nonzero native value reverts before retention, refund, or other work.
  - ☐ Negative test
- Pause or approval failure prevents principal pulls, fees, and routing.
  - ☐ Negative test
- Arithmetic, cast, dependency, malformed-return, or swap failures revert atomically.
  - ☐ Negative test
- Direct TIP20 `transferFrom`, `approve`, and `transfer` boolean results are ignored.
  - ☐ Negative test
- Zero recipients, unsupported EIDs, zero hops, and zero cleaned amounts lack rejection.
  - ☐ Negative test
- Fee-token spending lacks a caller-specified maximum.
  - ☐ Negative test
- Trusted dependencies may reenter, but eventual reverts roll back effects.
  - ☐ Negative test

Function call analysis

- `this.sendOFT(...) -> this.removeDust(...) -> IOFT2(_oft).decimalConversionRate()`
- `this.sendOFT(...) -> IOFT(_oft).token() -> ITIP20(...).transferFrom(...)`
- `this.sendOFT(...) -> IOFT(_oft).quoteSend(...) -> TIP_FEE_MANAGER.userTokens(...)`
- `this.sendOFT(...) -> STABLECOIN_DEX.swapExactAmountOut(...) -> IOFT(_oft).send{value: 0}(...)`
- External edges propagate reverts; TIP20/send return values remain ignored, excess swap input is refunded to the original caller, and quote sizing remains unchanged.
- **Manual call-path verification**: ASI adds self-calls, base operations, composer callbacks, and a second dust-clean that are absent from source. Replay-identity collisions and queued-settlement custody changes are confined to inbound `lzCompose`.

Function: `setApprovedOft(address _oft, bool _isApproved)`

This admin-only setter writes `_isApproved` directly to `HopV2Storage.approvedOft[_oft]` in the fixed ERC-7201 namespace. It validates neither the address nor pending claims and emits no event or return data.

Inputs

- `_oft`
  - **Control**: Authorized calldata selects the mapping key, while authorization depends on `msg.sender` rather than `tx.origin`.
  - **Constraints**: Any canonically encoded address is accepted, including zero, EOAs, and incompatible contracts, but `msg.value` must be zero.
  - **Impact**: The address is stored only as a key, although later consumers may treat an approved key as an OFT.
- `_isApproved`
  - **Control**: An account holding `DEFAULT_ADMIN_ROLE` chooses the boolean value.
  - **Constraints**: The value must use canonical ABI encoding and may equal the currently stored bit.
  - **Impact**: The value is stored unchanged; `true` admits later approval gates, while `false` causes `InvalidOFT` after preceding checks.

Branches and code coverage

Intended branches

- An authorized call writes `false` to the exact approved-OFT key, which the downstream wrapper gate reads.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_UnapprovedOft_QuoteAndSendFailCl`
- Setting `true` permits the key to pass later send, compose, and wrapper approval gates, subject to their other checks.
  - ☐ Test coverage
- Setting approval false blocks the later wrapper quote and send gates.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_UnapprovedOft_QuoteAndSendFailCl`
- Deferral of already credited compose settlement remains untested.
  - ☐ Test coverage
- Reassigning the existing value succeeds without a distinct branch, event, external

interaction, or return value.

- ☐ Test coverage

Negative behavior

- Malformed calldata, including a noncanonical boolean, reverts before authorization or storage changes.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point before the body executes.
  - ☐ Negative test
- A caller lacking `DEFAULT_ADMIN_ROLE` reverts before the storage pointer is obtained or written.
  - ☐ Negative test
- Zero, EOA, or incompatible `_oft` addresses are intentionally accepted, with failures deferred to later OFT use.
  - ☐ Negative test
- Deapproval does not reconcile queued compose claims and can defer settlement until trusted administration restores approval.
  - ☐ Negative test
- No supplied evidence directly covers this setter's authorization, assignments, entry failures, or queued-claim case.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
  - ASI's `this.` prefix is incorrect. Source inspection confirms an internal fixed-slot pointer assignment with no external call, controllable return, reentrancy, or explicit revert path despite unavailable Yul analysis.
- **Manual call-path verification**: The path is nonpayable ABI entry, inherited role authorization, internal fixed-slot pointer construction, and one mapping write. Any later revert rolls it back atomically, and downstream OFT, token, fee, precompile, or delivery interactions occur only in separate transactions.

Function: `setExecutorOptions(uint32 eid, bytes _options)`

Holders of `DEFAULT_ADMIN_ROLE` can store arbitrary executor-option bytes by EID in `HopV2Storage`, replacing prior data. The call emits nothing, moves no value or tokens, and later affects `quoteHop` fee calculation.

Inputs

- `eid`
  - **Control**: Any caller can encode the key, but only `DEFAULT_ADMIN_ROLE` may update it.
  - **Constraints**: ABI decoding enforces `uint32`, with no destination or nonzero validation.
  - **Impact**: Selects the mapping entry later read by `quoteHop` for the same destination EID.
- `_options`
  - **Control**: An authorized caller may supply arbitrary dynamic bytes.
  - **Constraints**: No format, length, version, or nonemptiness validation is applied.
  - **Impact**: Replaces the stored bytes; empty bytes select later defaults, while nonempty bytes may gain compose options and influence or revert executor quoting.

Branches and code coverage

Intended branches

- Authorized success path: an authorized call stores `_options` at `executorOptions[eid]` without emitting an event or returning data.
  - ☐ Test coverage
- All option values follow the same body path: empty and nonempty bytes use the same assignment path, with their distinction deferred to `quoteHop`.
  - ☐ Test coverage
- Replacement and clearing behavior: nonempty bytes overwrite prior data, while empty bytes clear the logical value readable through `executorOptions(eid)`.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller lacking `DEFAULT_ADMIN_ROLE` reverts before `HopV2Storage` access or assignment and preserves the prior value; `onlyRole` still reads access-control storage.
  - ☐ Negative test
- Native value supplied to the external entry point: because the function is nonpayable, nonzero `msg.value` reverts before modifier or body execution.
  - ☐ Negative test
- Invalid ABI encoding: malformed dynamic-bytes calldata reverts during ABI decoding

before state mutation.

- ☐ Negative test

- Resource exhaustion: memory, calldata-copying, storage, or gas failure reverts atomically without a partial replacement.
  - ☐ Negative test
- No option-validation rejection exists: once authorized, any decodable EID and option bytes are accepted, with downstream interpretation failures deferred to later consumers.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
  - The generated `this.*` trace is incorrect. The source calls an internal pure helper whose Yul only assigns the fixed ERC-7201 storage slot, cannot reenter, and makes no external call.
- **Manual call-path verification**: The immediate path covers nonpayable and ABI checks, `onlyRole`, fixed-slot lookup, and one atomic mapping assignment. Later quotes call configured DVN, executor, and treasury contracts by `STATICCALL`. Tempo sends use trusted callback-capable OFT, TIP20, FeeManager, and DEX paths, and failures revert all effects. Direct-send fee-token bounds and quote sizing are separate concerns. Dust can arise only from future rate changes because the current rate is one.

Function: `setFeeMultipliers(uint32 _eid, uint64 _dvn, uint64 _executor, uint64 _treasury)`

An authorized administrator atomically overwrites one EID's DVN, executor, and treasury multipliers without events, returns, transfers, or external calls. Zero preserves raw fees, while nonzero values scale by `multiplier / 10_000` with downward rounding; EIDs and multipliers are otherwise unchecked.

Inputs

- `_eid`
  - **Control**: Calldata supplies the mapping key, subject to `DEFAULT_ADMIN_ROLE` authorization through `msg.sender`.
  - **Constraints**: ABI decoding enforces `uint32`, but unsupported, local, remote, and zero EIDs are accepted unchanged.
  - **Impact**: Selects the complete multiplier entry and later becomes the EID passed to the DVN, executor, and treasury fee services.

- `_dvn`
  - **Control**: A `DEFAULT_ADMIN_ROLE` holder supplies this value in calldata.
  - **Constraints**: ABI decoding enforces `uint64`, but the value is uncapped and stored unchanged.
  - **Impact**: Zero preserves the raw DVN fee, while nonzero values scale it before multiplication by `numDVNs` and inclusion in the treasury input.
- `_executor`
  - **Control**: A `DEFAULT_ADMIN_ROLE` holder supplies this value in calldata.
  - **Constraints**: ABI decoding enforces `uint64`, but the value is uncapped and stored unchanged.
  - **Impact**: Zero preserves the raw executor fee, while nonzero values scale it before inclusion in the treasury input and final hop fee.
- `_treasury`
  - **Control**: A `DEFAULT_ADMIN_ROLE` holder supplies this value in calldata.
  - **Constraints**: ABI decoding enforces `uint64`, but the value is uncapped and stored unchanged.
  - **Impact**: Zero preserves the raw treasury fee, while nonzero values scale it after the treasury call and affect the final collected fee.

The caller (`msg.sender`) controls authorization, `tx.origin` is unread, and malformed calldata or nonzero `msg.value` is rejected before mutation.

Branches and code coverage

Intended branches

- An authorized call stores all three supplied multiplier values for the selected EID.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- The fixed storage-slot resolution is not independently instrumented.
  - ☐ Test coverage
- Unset zero multipliers preserve raw fee components, while an explicit `10_000` setting leaves the quote unchanged.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- Stored nonzero `20_000` multipliers increase the resulting quote and are returned by the getter.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- A nonintegral case that proves rounding down remains untested.

- ☐ Test coverage

- The tested key begins unset, accepts a `10_000` struct, and is then completely replaced by `20_000` values.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- Absence of events and ABI return data is not asserted.
  - ☐ Test coverage
- RemoteHopV201Tempo inherits the setter and `quoteHop`; changing the stored multipliers changes its quote.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- Resolution of other virtual hooks is not exercised by this test.
  - ☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE` reverts.
  - ☑ Negative test (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers_NotAdm`
- Reversion before storage resolution or mutation is not independently asserted.
  - ☐ Negative test
- A call carrying nonzero `msg.value` is rejected by the nonpayable entry point before execution.
  - ☐ Negative test
- Malformed calldata or invalid upper bits for `uint32` or `uint64` revert during ABI decoding.
  - ☐ Negative test
- Unsupported EIDs, zero multipliers, and large ABI-valid multipliers are accepted administrator configurations rather than revert branches.
  - ☐ Negative test
- Later service, decoding, or checked-arithmetic failures revert that quote without undoing the committed configuration transaction.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`

    - Source verification shows an internal pure Yul helper that selects the fixed HopV2 storage slot and cannot call out, reenter, transfer value, or be redirected by runtime input.
- **Downstream stored-value interaction verification**: The `quoteHop` function staticcalls `DVN.getFee`, `EXECUTOR.getFee`, and `TREASURY.getFee`. Later sends may call fee-token, DEX, FeeManager, and `IOFT` dependencies, with failures atomically rolling back pulls, swaps, approvals, sends, and state changes.
- **Manual call-path verification**: The setter performs authorization, fixed-slot resolution, and one struct overwrite. ASI's `this._getHopV2Storage()` trace is incorrect because the helper is internal and pure. Direct hop calls still lack a caller-bound fee token and input cap, while wrapper calls are non-reentrant and gross-bounded. Zero-recipient handling, quote sizing, replay identity, and queued-settlement custody are independent runtime concerns.

Function: `setFeeMultipliersBatch(uint32[] _eids, FeeMultipliers[] _multipliers)`

This admin-only nonpayable function stores DVN, executor, and treasury multipliers by EID for inherited `quoteHop`; duplicate EIDs are last-write-wins, and empty batches do nothing. Administrators and configured fee services are trusted.

Inputs

- `_eids`
    - **Control**: An authorized caller selects each `uint32` mapping key.
    - **Constraints**: The array must be well-formed and equal in length to `_multipliers`, but EIDs need not be supported, nonzero, remote, or unique.
    - **Impact**: Entries are written in order, so duplicate EIDs retain the last supplied multiplier and an empty array changes nothing.
- `_multipliers`
    - **Control**: An authorized caller selects each complete `FeeMultipliers` struct.
    - **Constraints**: Each component is `uint64`; zero means unscaled, `10_000` means 1x, and no application-level maximum is enforced.
    - **Impact**: The `quoteHop` function scales DVN and executor fees before treasury pricing, then scales treasury fees before applying the separately configured `hopFee`. Extreme values may make checked arithmetic revert.

Branches and code coverage

Intended branches

- Source-reviewed authorization success: a caller with `DEFAULT_ADMIN_ROLE` proceeds

using `msg.sender`, not `tx.origin`.

☐ Test coverage

- Source-reviewed equal-length, empty batch: equal empty arrays pass validation and perform no writes.

☐ Test coverage

- An equal-length nonempty batch writes complete structs by index.

☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers_Batch`)

- Duplicate-EID last-write behavior remains untested.

☐ Test coverage

- Successful execution commits both tested batch entries.

☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers_Batch`)

- ABI return data is not asserted.

☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE` reverts for the batch setter.

☑ Negative test (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers_NotAdm`

- Ordering before length validation and storage writes is not instrumented.

☐ Negative test

- Source-reviewed malformed calldata: ABI decoding rejects malformed arrays, structs, offsets, lengths, or out-of-width values before writes.

☐ Negative test

- Unequal array lengths revert with `LengthMismatch()`.

☑ Negative test (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers_Batch_I`

- Reversion before storage-pointer resolution and writes is not independently asserted.

☐ Negative test

- Source-reviewed nonpayable boundary: nonzero `msg.value` is rejected before the body executes.

☐ Negative test

- Source-reviewed resource or arithmetic failure: out-of-gas or any other failure reverts the entire batch without partial writes.

☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
    - ASI renders an external self-call that does not occur. Source review shows an internal pure Yul assignment to the fixed `HopV2Storage` namespace with no external interaction or caller-controllable base slot.
- **Manual call-path verification**: The path is nonpayable/ABI checks, internal role authorization, length validation, fixed-slot acquisition, and ordered writes. Later, `quoteHop` uses `STATICCALL`s to configured fee services, whose failures revert, while direct-send rollback reverses prior `transferFrom` and wrappers quote before collecting funds.

Function: `setHopFee(uint256 _hopFee)`

An authorized default administrator stores any `uint256` markup without validation. Downstream fee-bearing routes apply it after component fees, so extreme values may overflow or fail Tempo `uint128` casts.

Inputs

- `_hopFee`
    - **Control**: The authorized caller selects the exact stored markup.
    - **Constraints**: Any `uint256` is accepted, although later addition, multiplication, or `uint128` conversion may revert.
    - **Impact**: It overwrites `HopV2Storage.hopFee` and scales downstream fee aggregates by `(10_000 + hopFee) / 10_000`.
- `msg.sender`
    - **Control**: It is the immediate EOA or contract caller.
    - **Constraints**: It must hold `DEFAULT_ADMIN_ROLE`.
    - **Impact**: It affects only authorization and is not stored, emitted, or used as a call target.
- `msg.value`
    - **Control**: The caller can attempt to attach native value.
    - **Constraints**: The nonpayable boundary requires `msg.value == 0`.
    - **Impact**: It is neither read nor forwarded, and value-bearing calls revert before mutation.
- `tx.origin`
    - **Control**: It is an ambient transaction property not read by this function.
    - **Constraints**: No authorization or validation depends on it.
    - **Impact**: It has no effect on storage, fees, transfers, or call targets.

Branches and code coverage

Intended branches

- An authorized zero-value call overwrites `HopV2Storage.hopFee` without events, transfers, or return data.
  - ☐ Test coverage
- Every ABI-valid `_hopFee`, including `0`, `10_000`, and `type(uint256).max`, can be stored without a range check.
  - ☐ Test coverage
- A successful fee-bearing `quoteHop` applies the stored markup after the configured component fees.
  - ☐ Test coverage
- Local or Fraxtal-related RemoteHopV201Tempo routes set the second-leg hop fee to zero.
  - ☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE` reverts before storage mutation.
  - ☐ Negative test
- A call with `msg.value > 0` reverts at the nonpayable boundary.
  - ☐ Negative test
- A stored value above `type(uint256).max - 10_000` makes later fee-bearing quotes revert during addition.
  - ☐ Negative test
- A sufficiently high markup can make later fee multiplication overflow while leaving the earlier stored configuration intact.
  - ☐ Negative test
- A marked-up Tempo fee exceeding `uint128` reverts when downstream conversion requires `SafeCast.toUint128`.
  - ☐ Negative test

The setter body has no conditional branch beyond inherited role enforcement and the compiler-enforced nonpayable boundary.

Function call analysis

- `this._getHopV2Storage()`

- The generated `this.` qualification is incorrect: `_getHopV2Storage()` is an internal pure fixed-slot helper with no callback, transfer, arithmetic, cast, or reentrancy.

- **Manual call-path verification**: The setter makes no external calls. Downstream quotes call trusted DVN, executor, and treasury providers. Remote Tempo atomically collects TIP20 fees, performs the configured exchange/OFT calls, returns zero, and bypasses `_handleMsgValue`. Direct hop users have no caller-selected maximum fee-token debit, while TempoFeeInclusiveWrapper bounds gross spend and minimum net delivery.

Function: `setMessageProcessed(address _oft, uint32 _srcEid, uint64 _nonce, byte[32] _composeFrom)`

Holders of `DEFAULT_ADMIN_ROLE` can set the four-field replay key and emit `MessageHash`, even if it is already set. Exact-tuple pre-marking can cause a later compose to finalize without payout. Distinct authenticated GUIDs can collide because the key omits the GUID, and the generated `byte[32]` header differs from the source `bytes32` type.

Inputs

- `_oft`
  - **Control**: The authorized caller selects this calldata value.
  - **Constraints**: It is an address with no existence, approval, trust, or nonzero validation.
  - **Impact**: It selects the first replay-key component and is emitted in `MessageHash`.
- `_srcEid`
  - **Control**: The authorized caller selects this calldata value.
  - **Constraints**: It is ABI-limited to `uint32` but needs not identify a supported or configured chain.
  - **Impact**: It selects the second replay-key component and is emitted as `srcEid`.
- `_nonce`
  - **Control**: The authorized caller selects this calldata value.
  - **Constraints**: It is ABI-limited to `uint64` with no existence, freshness, monotonicity, or prior-state check.
  - **Impact**: It selects the third replay-key component, and repeated calls still emit `MessageHash`.
- `_composeFrom`
  - **Control**: The authorized caller selects this calldata value.
  - **Constraints**: The source type is `bytes32`, with no nonzero or configured-remote-hop validation.

    - **Impact**: It selects the fourth replay-key component and can cause matching `lzCompose` execution to return before payout.
- `msg.sender`
    - **Control**: The immediate caller determines this identity.
    - **Constraints**: It must hold inherited `DEFAULT_ADMIN_ROLE`.
    - **Impact**: It controls authorization but is absent from the replay key and event.
- `msg.value`
    - **Control**: The caller chooses the transaction value.
    - **Constraints**: Any nonzero value reverts at nonpayable dispatch.
    - **Impact**: No native value is accepted, moved, or retained.
- `tx.origin`
    - **Control**: The transaction originator determines this value.
    - **Constraints**: It is never read.
    - **Impact**: It has no effect.

Branches and code coverage

Intended branches

- Authorized execution and pre-marking: an authorized call sets the selected key to `true` and emits `MessageHash`, and the removed fork probe showed exact-tuple pre-marking finalized compose without payout.
    - ☐ Test coverage
- Fresh-entry state effect: a previously false entry becomes true with no mechanism here to restore it.
    - ☐ Test coverage
- Already-processed state effect: an already true entry remains true and another `MessageHash` is emitted without revert.
    - ☐ Test coverage
- Concrete downstream duplicate path: a later matching `lzCompose` returns before `_sendLocal` when its exact tuple has been pre-marked, leaving separately credited principal in the hop.
    - ☐ Test coverage
- GUID-independent collision path: distinct authenticated GUIDs sharing the four local fields collide, as shown by the supplied natural-validator reproduction.
    - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller lacking `DEFAULT_ADMIN_ROLE` reverts before hashing,

storage mutation, or event emission.

- ☐ Negative test

- Nonzero native value: nonzero `msg.value` reverts at dispatch before modifier or body execution.
  - ☐ Negative test
- No semantic tuple validation: authorized callers may submit arbitrary zero, unsupported, unapproved, future, or nonexistent tuple values.
  - ☐ Negative test
- No GUID binding: the API cannot distinguish different GUIDs that share the same four-field tuple.
  - ☐ Negative test
- Atomic local failure: any local revert discards the mapping update and event, while later nonduplicate `_sendLocal` or `hopCompose` failure rolls back validator marking and permits retry.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
  - The generated `this.` trace is inaccurate because `_getHopV2Storage()` is an internal pure Yul helper that binds a fixed storage slot without calls, callbacks, value movement, or reentrancy.
- **Manual call-path verification**: Execution performs nonpayable dispatch, role checking, fixed-slot lookup, `abi.encode`, hashing, storage assignment, and event emission only. Matching duplicates return before later token or composer interactions, while nonduplicate interactions occur solely during endpoint-controlled `lzCompose`.

Function: `setNumDVNs(uint32 _numDVNs)`

This role-gated setter writes any `uint32`, including zero and the maximum, to `HopV2Storage.numDVNs` without validation, events, returns, transfers, or external calls. The stored count later scales `quoteHop`'s DVN fee and downstream fee accounting.

Inputs

- `_numDVNs`
  - **Control**: The immediate caller selects the value, but only `msg.sender` holding `DEFAULT_ADMIN_ROLE` can store it.

- **Constraints**: ABI decoding limits the value to `uint32`, while zero and `type(uint32).max` are accepted and nonzero `msg.value` is rejected.
- **Impact**: The exact value is stored at the fixed ERC-7201 slot and returned by `numDVNs()`, later scaling quoted DVN fees and related collection, refund, and wrapper accounting.

Branches and code coverage

Intended branches

- An authorized zero-value call stores `_numDVNs` and returns empty data.
  - ☐ Test coverage
- Every ABI-valid value, including zero and `type(uint32).max`, follows the same unchecked storage-write path.
  - ☐ Test coverage
- The RemoteHopV201Tempo contract inherits and executes this setter unchanged.
  - ☐ Test coverage
- The internal `_getHopV2Storage()` helper assigns the fixed storage slot without contract interaction.
  - ☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE` reverts before storage mutation.
  - ☐ Negative test
- A call carrying nonzero `msg.value` reverts before authorization or storage mutation.
  - ☐ Negative test
- Zero and large ABI-valid values do not revert, though downstream checked fee arithmetic may later revert.
  - ☐ Negative test

Function call analysis

- `this._getHopV2Storage()`
  - The generated `this._getHopV2Storage()` trace is incorrect. The source uses an internal pure Yul assignment of the fixed `HopV2StorageLocation`, with no external call, callback, controllable return, or reentrancy.
- **Manual call-path verification**: The direct path is `setNumDVNs` → inherited `onlyRole(DEFAULT_ADMIN_ROLE)` → internal `_getHopV2Storage()` → `numDVNs` assignment. Downstream `quoteHop` calls trusted configured DVN, executor, and

treasury contracts. Any uncaught quote, collection, send, refund, callback, or arithmetic failure rolls back its enclosing transaction.

Function: `setRemoteHop(uint32 _eid, address _remoteHop)`

Restricted to `DEFAULT_ADMIN_ROLE`, this setter zero-left-pads `_remoteHop` to `bytes32` and stores it at `remoteHop[_eid]` without validation or an event. The configuration governs inbound trusted-hop metadata, while only `FRAXTAL_EID` (`30_255`) affects concrete outbound routing.

Inputs

- `_eid`
  - **Control**: An authorized administrator selects the mapping key.
  - **Constraints**: ABI decoding enforces `uint32`, but the function performs no EID or settlement-safety validation.
  - **Impact**: Any EID affects matching inbound classification, while only `FRAXTAL_EID` affects concrete outbound routing.
- `_remoteHop`
  - **Control**: An authorized administrator selects the address value.
  - **Constraints**: ABI decoding enforces `address`, which is preserved and zero-left-padded to `bytes32` without further validation.
  - **Impact**: Zero, codeless, unchanged, unsupported, or settlement-time replacement values are stored exactly and may affect trust metadata or routing.

Branches and code coverage

Intended branches

- The address overload converts and writes the Fraxtal hop used by a successful multihop send.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_ToChain`)
- The write may create, overwrite, repeat, or zero an entry without a source-level branch.
  - ☐ Test coverage
- The transaction origin (`tx.origin`) is unused, while nonzero `msg.value` is rejected before execution because the function is nonpayable.
  - ☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE`, including one holding only `PAUSER_ROLE`, reverts before storage changes.
  - ☐ Negative test
- Malformed or noncanonical calldata fails ABI decoding before authorization or storage execution.
  - ☐ Negative test
- Invalid, zero, codeless, unchanged, unsupported, or settlement-time configurations are accepted rather than rejected.
  - ☐ Negative test

Function call analysis

- `this._setRemoteHop(_eid, byte[32](uint256(uint160(_remoteHop)))) -> this._getHopV2Storage()`
  - Both calls are internal; authorization precedes the sole storage write, and the Yul helper only binds the fixed `HopV2StorageLocation` slot.
- **Manual call-path verification**: The trace's `this.` prefixes and `byte[32]` rendering are inaccurate, and it omits `onlyRole`. No external call, callback, or reentrancy point exists in the setter. Downstream, inbound compose compares `remoteHop[srcEid]` before token transfer or callback, while routed/compose outbound quote and send paths consume `remoteHop(FRAXTAL_EID)`; plain empty-data Fraxtal transfers do not. Rotation during separately queued compose settlement can alter delivered source/sender metadata and make strict recipients revert until restoration, while an incorrect `FRAXTAL_EID` value can misroute later sends.

Function: `setRemoteHop(uint32 _eid, byte[32] _remoteHop)`

This inherited nonpayable administrator setter replaces `remoteHop[_eid]`; the generated `byte[32]` header is preserved although the source type is `bytes32`. The mapping classifies inbound metadata and, for `FRAXTAL_EID`, supplies future routed destinations.

Inputs

- `_eid`
  - **Control**: The authorized caller supplies the mapping key unchanged.
  - **Constraints**: Any canonically decoded `uint32` is accepted without source-level validation.
  - **Impact**: Only `FRAXTAL_EID` (`30_255`) affects concrete outbound routing, while any key can affect inbound classification.
- `_remoteHop`

- **Control**: The authorized caller supplies the stored `bytes32` value unchanged.
- **Constraints**: Any `bytes32`, including zero or a noncanonical address-shaped value, is accepted.
- **Impact**: The value controls inbound source/sender trust and can become the future Fraxtal `SendParam.to`.

Branches and code coverage

Intended branches

- The inherited `bytes32` setter stores the configured Fraxtal peer used by a successful multihop send.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_ToChain`)
- A same-value overwrite is not isolated.
  - ☐ Test coverage
- Unrestricted decoded key and value: all canonically decoded key/value combinations follow the same write path.
  - ☐ Test coverage — no supplied test is mapped specifically to zero, local-EID, unused-EID, same-value, or noncanonical-address–shaped inputs for this overload
- Inbound trust rotation: rotation can rewrite callback identity and make a strict recipient revert until restoration, without changing recipient or amount.
  - ☐ Test coverage — the supplied removed fork probe is mapped to remote-hop rotation changing callback identity, causing a strict recipient to reject, and restoration of the old mapping permitting the prior identity again
- Setting `remoteHop(FRAXTAL_EID)` supplies the routed multihop destination used by a successful send.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_ToChain`)
- A mutation test contrasting local and direct-Fraxtal branches is absent.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller lacking `DEFAULT_ADMIN_ROLE` reverts before storage changes.
  - ☐ Negative test — no exact supplied test is mapped to this setter's authorization-failure boundary
- Native value supplied: nonzero `msg.value` reverts at the nonpayable boundary.

- ☐ Negative test — no exact supplied test is mapped to this setter's nonpayable boundary

- Malformed ABI input: noncanonical `(uint32,bytes32)` calldata reverts before authorization and execution.
  - ☐ Negative test — no exact supplied test is mapped to malformed or out-of-range ABI calldata for this overload
- Invalid peer data is not rejected: authorized zero, stale, unintended, or noncanonical peer values are stored without validation.
  - ☐ Negative test — no exact supplied test is mapped specifically to acceptance of these values, and the source contains no validation-revert branch for them

Function call analysis

- `this._setRemoteHop(_eid, _remoteHop) -> this._getHopV2Storage()`
  - These generated `this.*` labels represent internal calls, and the Yul helper only binds the fixed namespaced storage slot before one mapping assignment.
- **Manual call-path verification**: The setter makes no external call, transfer, callback, or reentrant edge. Later approved OFT quote and send calls consume Fraxtal routing. Data-bearing inbound delivery may call the decoded recipient, and a callback revert rolls back that compose attempt but not the prior OFT credit or committed configuration.

5.21. Module: LinearRewardsErc4626_2.sol

Function: `constructor(ERC20 _underlying, string _name, string _symbol)`

This abstract base constructor configures ERC4626 metadata and immutables for SfrxUSD2/SfrxUSD3/SfrxUSD; FrxUSD cannot reach it. It requires the first `decimals()` response to be 18, derives precision from the second, converts `ONE_YEAR`, and itself moves or emits nothing.

Inputs

- `_underlying`
  - **Control**: The deployer selects the underlying address, which is forwarded to ERC4626 and queried twice.
  - **Constraints**: Base construction and both calls must succeed, the first result must equal 18, and the second exponent must not exceed 77.
  - **Impact**: The second result determines immutable `UNDERLYING_PRECISION`, while zero, no-code, incompatible, or context-dependent targets may revert or return differing values.

- `_name`
    - **Control**: The deployer selects it for SfrxUSD2, while SfrxUSD3 and SfrxUSD fix it to `"Staked Frax USD"`.
    - **Constraints**: It is forwarded unchanged to ERC4626 without local content validation.
    - **Impact**: It configures inherited metadata in the implementation creation context rather than proxy storage.
- `_symbol`
    - **Control**: The deployer selects it for SfrxUSD2, while SfrxUSD3 and SfrxUSD fix it to `"sfrxUSD"`.
    - **Constraints**: It is forwarded unchanged to ERC4626 without local content validation.
    - **Impact**: It configures inherited metadata in the implementation creation context rather than proxy storage.

Branches and code coverage

Intended branches

- Concrete `SfrxUSD2` constructor path: deployment forwards all vault arguments, completes base construction, and then sets `_initialized = true`.
    - ☐ Test coverage
- Concrete `SfrxUSD3` constructor path: deployment forwards `_underlying` while fixing the name, symbol, and timelock address.
    - ☐ Test coverage
- Concrete `SfrxUSD` constructor path: deployment forwards `_underlying` through SfrxUSD3 without an additional constructor body.
    - ☐ Test coverage
- Successful local decimal validation and precision derivation: the first result equals 18, and the second fitting exponent initializes `UNDERLYING_PRECISION`.
    - ☐ Test coverage
- Constant year conversion: internal `convert(ONE_YEAR)` initializes immutable `ONE_YEAR_UD60X18` from `31_536_000`.
    - ☐ Test coverage
- Base-constructor completion: ERC4626 construction completes before either explicit local decimals call.
    - ☐ Test coverage

Negative behavior

- First local result is not 18: a revert with `UnderlyingAssetMustBe18Decimals()` aborts creation and rolls back construction effects.
  - ☐ Negative test
- First local static-call or decode failure: a revert, gas exhaustion, or malformed result aborts creation.
  - ☐ Negative test
- Second local static-call, decode, or arithmetic failure: call or decode failure reverts, while an exponent of at least 78 overflows checked exponentiation.
  - ☐ Negative test
- Imported or descendant constructor failure: any ERC4626, Timelock2Step, or later constructor revert aborts and rolls back creation.
  - ☐ Negative test
- Native-value rejection: nonzero `msg.value` fails because the constructor chain is nonpayable.
  - ☐ Negative test
- No frxUSD path: FrxUSD, FrxUSD2, and FrxUSD3 do not inherit this constructor.
  - ☐ Negative test

Function call analysis

- `ERC4626(_underlying, _name, _symbol)`
  - This same-context base construction must complete before the local body, but the imported Solmate implementation is not source verified.
- First `_underlying.decimals()` in `if (_underlying.decimals() != 18)`
  - This zero-value `STATICCALL` must decode to 18, or deployment reverts atomically.
- Second `_underlying.decimals()` in `10 ** _underlying.decimals()`
  - This independent zero-value `STATICCALL` supplies the checked exponent, so context-dependent results can produce non-1e18 precision or overflow.
- `convert(ONE_YEAR)`
  - This internal conversion consumes only the fixed `31_536_000` constant and initializes `ONE_YEAR_UD60X18`.
- `LinearRewardsErc4626_2(ERC20(address(_underlying)), _name, _symbol)` from SfrxUSD2
  - This same-context chain preserves the address bits and reaches `_initialized = true` only after successful base construction.
- `SfrxUSD2(IERC20(_underlying), "Staked Frax USD", "sfrxUSD", address(0))`

from SfrxUSD3

- This same-context chain forwards the selected underlying with fixed metadata and timelock arguments.

- `SfrxUSD3(_underlying)` from SfrxUSD
  - This final concrete path forwards the selected underlying without additional constructor behavior.
- **Manual call-path verification**: The resolved SfrxUSD2/SfrxUSD3/SfrxUSD paths contain two explicit decimals staticcalls and one internal conversion; immutables and constructor writes affect implementation creation rather than proxy storage, with correct deployment configuration trusted to governance.

Function: `deposit(uint256 _assets, address _receiver)`

This deprecated function is inherited unchanged by the scoped sfrxUSD contracts and disabled for every caller. Reaching its body immediately reverts with `MintRedeemsDisabled()` without state changes, asset movement, share minting, or successful return.

Inputs

- `_assets`
  - **Control**: The caller controls this value, and no scoped internal call site supplies it.
  - **Constraints**: It is type-constrained to `uint256`, while ABI-valid calldata and zero call value are required to reach the body.
  - **Impact**: It is unused and cannot affect storage, arithmetic, transfers, minting, calls, events, or return data.
- `_receiver`
  - **Control**: The caller controls this address, and no scoped internal call site supplies it.
  - **Constraints**: It must ABI-decode as an address, but zero, caller, vault, and arbitrary addresses are otherwise unchecked.
  - **Impact**: It is unused and cannot become a call target, transfer recipient, share recipient, stored value, event field, or return value.

Branches and code coverage

Intended branches

- Source-reviewed disabled path: every invocation reaching the body immediately reverts with `MintRedeemsDisabled()`, with no successful state-changing or return branch.

- ☐ Test coverage

- External nonpayable boundary: nonzero-value direct calls or implementation delegatecalls fail Solidity's nonpayable entry check before reaching the body.
  - ☐ Test coverage
- Zero-value call modes: ABI-valid ordinary calls, delegatecalls, and static invocations with zero value reach the body and revert with `MintRedeemsDisabled()`.
  - ☐ Test coverage
- Internal-call behavior: any hypothetical internal call would encounter the same unconditional revert, although no scoped internal call site exists.
  - ☐ Test coverage

Negative behavior

- Untrusted, privileged, EOA, and contract callers reaching the body receive `MintRedeemsDisabled()` rather than minted shares.
  - ☐ Negative test
- Boundary values for `_assets` and `_receiver` do not create a successful path.
  - ☐ Negative test
- Malformed selector or argument calldata fails dispatch or ABI decoding before the body and need not return `MintRedeemsDisabled()`.
  - ☐ Negative test
- The reported 73/73 mainnet compliance/upgrade tests and 15/15 sfrxUSD signature tests do not establish exact coverage of these branches.
  - ☐ Negative test

This entry point remains disabled regardless of minter administration, freeze state, timelock configuration, or proxy-deployment assumptions.

Function call analysis

- **Manual call-path verification**: Source review confirms no internal dispatch, hooks, callbacks, token operations, reentrancy points, or alternate scoped implementation; callers may catch the revert only across an external-call boundary, without creating a successful `deposit` execution.

Function: `depositWithSignature(uint256 _assets, address _receiver, uint256 _deadline, bool _approveMax, uint8 _v, byte[32] _r, byte[32] _s)`

Every zero-value, ABI-valid call that reaches the function reverts with `MintRedeemsDisabled()`, producing no return, state change, event, asset movement, or external interaction. The source uses `bytes32` rather than the generated `byte[32]`; the unreachable functions and trust-boundary concerns do not affect this behavior.

Inputs

- `_assets`
  - **Control**: The untrusted caller controls the full-width `uint256`.
  - **Constraints**: Any canonically encoded value is accepted by dispatch but remains unread before the revert.
  - **Impact**: It cannot affect conversion, storage, deposits, or share minting.
- `_receiver`
  - **Control**: The untrusted caller controls the address without signer or caller restrictions.
  - **Constraints**: ABI decoding requires a canonical address encoding, but the body performs no validation.
  - **Impact**: It is never used as a recipient, target, storage key, or event field.
- `_deadline`
  - **Control**: The untrusted caller controls the full-width `uint256`.
  - **Constraints**: The value is never compared with `block.timestamp`, so every decoded value reaches the same revert.
  - **Impact**: It does not gate execution or affect signatures, state, calls, or events.
- `_approveMax`
  - **Control**: The untrusted caller controls the boolean.
  - **Constraints**: Only canonical boolean encodings decode, and either valid value reaches the same revert.
  - **Impact**: It cannot select an approval amount or alter token movement.
- `_v`
  - **Control**: The untrusted caller controls the `uint8`.
  - **Constraints**: Canonical values from 0 through 255 decode without signature validation.
  - **Impact**: It is not hashed, recovered, stored, returned, emitted, or forwarded.
- `_r`
  - **Control**: The untrusted caller controls this arbitrary source-level `bytes32`.

    - **Constraints**: Decoding requires one complete word, with no signature-format or nonzero check.
    - **Impact**: It does not affect recovery, storage, calls, returns, or events.
- `_s`
    - **Control**: The untrusted caller controls this arbitrary source-level `bytes32`.
    - **Constraints**: Decoding requires one complete word, with no low-`s`, format, or nonzero check.
    - **Impact**: It does not affect recovery, storage, calls, returns, or events.

Branches and code coverage

Intended branches

- The canonical `depositWithSignature(uint256,address,uint256,bool,uint8,bytes32,bytes32)` selector, zero `msg.value`, and valid ABI data reach `MintRedeemsDisabled()` with no successful return.
    - ☐ Test coverage
- Nonzero `msg.value` triggers Solidity's nonpayable guard before the function body.
    - ☐ Test coverage

Negative behavior

- An unknown selector, including one derived from the generated `byte[32]` notation, does not dispatch here.
    - ☐ Negative test
- Insufficient calldata or noncanonical `address`, `bool`, or `uint8` encodings revert during dispatch or decoding.
    - ☐ Negative test
- No caller class can bypass the revert because neither `msg.sender` nor `tx.origin` is read and no authorization branch exists.
    - ☐ Negative test
- All decoded argument values, including expired deadlines and arbitrary signature fields, terminate with the same custom-error revert.
    - ☐ Negative test

Function call analysis

- **Manual call-path verification**: Source inspection confirms the nonvirtual implementation is inherited unchanged, performs only `revert`

`MintRedeemsDisabled()`, and cannot transfer control, move assets, permit reentrancy, or retain effects; callers that catch the failure may continue with their own earlier state.

Function: `mint(uint256 _shares, address _receiver)`

This deprecated ERC-4626 entry point immediately reverts with `MintRedeemsDisabled()`; neither SfrxUSD3 nor SfrxUSD overrides it. Separate privileged `minter_mint` functionality is outside this path, and reviewed vault-dependency behavior is assumed.

Inputs

- `_shares`
  - **Control**: The `_shares` parameter is an untrusted caller-controlled `uint256` with no authorization constraint.
  - **Constraints**: ABI decoding requires a complete word, but the body never reads or validates the value.
  - **Impact**: No value can affect minting, transfers, storage, events, interactions, or returns because execution always reverts.
- `_receiver`
  - **Control**: The `_receiver` parameter is an untrusted caller-controlled address that need not equal `msg.sender`.
  - **Constraints**: ABI decoding requires a complete canonical address word, but the body never reads or validates the address.
  - **Impact**: No shares or assets can be assigned to the address because execution always reverts.

Branches and code coverage

Intended branches

- Resolved unconditional disable path: every ABI-valid zero-value invocation immediately reverts with `MintRedeemsDisabled()`, so no live asset path or successful return exists.
  - ☐ Test coverage
- Nonpayable dispatch boundary: any invocation with nonzero `msg.value` reverts before entering the function body and retains no value or state change.
  - ☐ Test coverage

Negative behavior

- Malformed or short calldata, including a noncanonical `_receiver` word, fails ABI decoding before `MintRedeemsDisabled()` is reached.

- ☐ Negative test

- Unusual share values, receiver addresses, callers, origins, or privileged roles cannot bypass the revert, although separate `minter_mint` functionality remains unaffected.
  - ☐ Negative test
- No exact test-to-branch mapping was supplied for `mint`.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: Resolution is SfrxUSD → SfrxUSD3 → SfrxUSD2 → `LinearRewardsErc4626_2.mint`, whose immediate revert makes constructor interactions, token operations, callbacks, and unrelated modules unreachable.

Function: `redeem(uint256 _shares, address _receiver, address _owner)`

This deprecated public ERC-4626 entry point immediately reverts with `MintRedeemsDisabled()` for every ABI-valid, zero-value call reaching its body. No sfrxUSD conversion or reward accounting executes under the stated upgrade-only assumptions.

Inputs

- `_shares`
  - **Control**: An untrusted caller controls the full `uint256` value.
  - **Constraints**: ABI-valid values are ignored without balance, allowance, authorization, arithmetic, or range checks.
  - **Impact**: The value cannot affect state, events, calls, transfers, or returns before the revert.
- `_receiver`
  - **Control**: An untrusted caller controls the decoded address.
  - **Constraints**: Any ABI-valid address, including `address(0)`, reaches the same revert.
  - **Impact**: The address is never used as a recipient, target, stored value, or event field.
- `_owner`
  - **Control**: An untrusted caller controls the decoded address without proving ownership or delegated authority.
  - **Constraints**: Any ABI-valid address is ignored without balance, allowance, or caller comparisons.
  - **Impact**: The address cannot select an account, consume allowance, burn

shares, or affect any return.

Branches and code coverage

Intended branches

- Every ABI-valid, zero-value call entering the body immediately reverts with `MintRedeemsDisabled()` and never returns `_assets`.
  - ☐ Test coverage
- Malformed or insufficient calldata fails ABI decoding before the custom-error statement executes.
  - ☐ Test coverage
- Nonzero `msg.value` fails at the external nonpayable boundary, while zero value permits body entry.
  - ☐ Test coverage
- The caller (`msg.sender`) cannot change behavior, and `tx.origin` is never read.
  - ☐ Test coverage

Negative behavior

- Redemption cannot burn shares, consume allowances, write storage, emit events, or transfer underlying assets.
  - ☐ Negative test
- ERC-4626 boundary inputs such as zero shares, invalid balances or allowances, zero addresses, and contract receivers all reach the same body-level revert after decoding.
  - ☐ Negative test
- Nonzero value and malformed calldata may fail before body entry without exposing `MintRedeemsDisabled()`.
  - ☐ Negative test
- The argumentless custom-error selector may be caught or inspected upstream; uncaught propagation rolls back the call chain.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The SfrxUSD → SfrxUSD3 → SfrxUSD2 chain resolves to the immediate-revert `LinearRewardsErc4626_2.redeem`; the displaced Solmate implementation, constructor-only `_underlying.decimals()` call, token operations, callbacks, hooks, and reentrancy paths are unreachable.

Function: `sync()`

The permissionless, nonpayable `sync()` function computes PRBMath exponential growth from `block.timestamp` and stored pricing state, stores and returns the new price, and then updates `lastSync`; it emits no event. Callers influence timing, while timelock-gated setters may configure zero or extreme pricing values.

Branches and code coverage

Intended branches

- Zero elapsed time or a zero rate preserves `pricePerShareStored`, which is rewritten and returned with the current timestamp.
  - ☐ Test coverage
- Positive elapsed time applies exponential growth using the old stored rate and then stores and returns the result.
  - ☐ Test coverage
- Permissionless callers produce identical behavior because neither `msg.sender` nor `tx.origin` is read.
  - ☐ Test coverage
- The `setPricePerShareIncPerSecond(uint256)` function authenticates the timelock, checkpoints using the old rate, and then stores and emits the new rate.
  - ☐ Test coverage
- All arithmetic that can revert completes before the consecutive price and timestamp writes.
  - ☐ Test coverage

Negative behavior

- A direct call with nonzero `msg.value` reverts at the nonpayable boundary without storage changes.
  - ☐ Negative test
- A future `lastSync` causes checked subtraction to underflow and revert before writes.
  - ☐ Negative test
- Overflow in `pricePerShareIncPerSecond * elapsedTime` reverts before wrapping or storage writes.
  - ☐ Negative test
- An unsupported PRBMath `exp` input reverts atomically without updating price or timestamp.
  - ☐ Negative test

- A fixed-point `mul` failure reverts atomically without clamping, fallback, or partial checkpointing.
    - ☐ Negative test
- Aggregate suite totals provide no exact branch-specific evidence.
    - ☐ Negative test

Function call analysis

- `this._previewPricePerShare(block.timestamp) -> Casting.sol.wrap(this.pricePerShareIncPerSecond * _elapsedTime)`
    - This internal retyping operation receives the checked product of elapsed time and the timelock-configured rate, with any overflow reverting before writes.
- `this._previewPricePerShare(block.timestamp) -> Math.sol.exp(_exponentUD60_18)`
    - This in-process PRBMath operation derives growth deterministically and may revert when the configured exponent exceeds its supported range.
- `this._previewPricePerShare(block.timestamp) -> Casting.sol.unwrap(Math.sol.mul(Casting.sol.wrap(this.pricePerShareStored), _ePowUD60_18))`
    - These internal operations produce the raw price returned and stored by `sync()`, with nested failures propagating before state changes.
- `this._previewPricePerShare(block.timestamp) -> Math.sol.mul(Casting.sol.wrap(this.pricePerShareStored), _ePowUD60_18)`
    - Fixed-point multiplication uses stored price and computed growth without any external target, callback, or reentrancy surface.
- `this._previewPricePerShare(block.timestamp) -> Casting.sol.wrap(this.pricePerShareStored)`
    - This internal conversion preserves the timelock-configured or legacy-proxy price, including zero or extreme values.
- **Manual call-path verification**: The source confirms `_previewPricePerShare` is an internal nonvirtual call mislabeled by ASI as `this`, with no external edge; correct proxy operation assumes an initialized legacy proxy with preserved pricing and vault state and correct immutables, not a fresh empty proxy.

Function: `withdraw(uint256 _assets, address _receiver, address _owner)`

This intentionally disabled ERC-4626 entry point always reverts with `MintRedeemsDisabled()` after successful decoding and value checks. It reads no parameters, caller identity, or storage and reaches no additional behavior.

Inputs

- `_assets`
  - **Control**: The untrusted caller supplies `_assets`.
  - **Constraints**: Valid `uint256` ABI encoding is required, but the body applies no validation.
  - **Impact**: The value is unused and cannot affect assets, shares, storage, or control flow.
- `_receiver`
  - **Control**: The untrusted caller supplies `_receiver`.
  - **Constraints**: Canonical address encoding is required, but the body performs no receiver validation.
  - **Impact**: The address is unused and receives no assets.
- `_owner`
  - **Control**: The untrusted caller supplies `_owner` without proving ownership or authorization.
  - **Constraints**: Canonical address encoding is required, but the body performs no balance, allowance, or ownership validation.
  - **Impact**: The address is unused, and no shares are burned or allowances consumed.

Branches and code coverage

Intended branches

- After dispatch, decoding, and the nonpayable check, the body always reverts with `MintRedeemsDisabled()`.
  - ☐ Test coverage
- Every caller identity follows the same path because `msg.sender` is not read.
  - ☐ Test coverage
- Every ABI-decodable input combination reverts without accounting, transfers, state changes, events, calls, or returns.
  - ☐ Test coverage

Negative behavior

- Nonzero `msg.value` fails the generated nonpayable check before reaching the body.
  - ☐ Negative test
- Missing or noncanonical calldata fails ABI decoding before the custom-error revert.
  - ☐ Negative test

- Every ABI-valid zero-value call reverts with `MintRedeemsDisabled()`, regardless of values, balances, or allowances.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The inherited implementation remains unchanged in the scoped SfrxUSD2 → SfrxUSD3 → SfrxUSD chain, invokes no parent, hook, helper, token, native-value, or external call, and its revert rolls back the frame while low-level callers may only catch the failure.

5.22. Module: PancakeV3Plugin.sol

Function: `getAmountOut(address pool, address tokenIn, uint256 amountIn, bool zeroForOne)`

This permissionless, nonpayable quote function commits transient callback authentication, calls a caller-selected pool, and clears transient state before returning. Any caught 32-byte revert becomes the quote, while normal completion returns zero.

Inputs

- `pool`
  - **Control**: The untrusted caller selects the external-call target.
  - **Constraints**: No nonzero, code, factory, interface, or canonical-pool validation is performed.
  - **Impact**: The target controls callback behavior, revert data, side effects, and the resulting quote.
- `tokenIn`
  - **Control**: The untrusted caller supplies the token address.
  - **Constraints**: It is not checked against pool tokens but is committed and encoded into callback data.
  - **Impact**: It binds callback authentication without being queried, approved, or transferred.
- `amountIn`
  - **Control**: The untrusted caller supplies the nominal quote amount.
  - **Constraints**: The `amountIn.toInt256()` conversion requires `amountIn <= type(int256).max`.
  - **Impact**: It binds callback authentication and becomes the pool's signed `amountSpecified`.

- `zeroForOne`
  - **Control**: The untrusted caller selects the swap direction.
  - **Constraints**: It is not committed or checked against `tokenIn` or pool ordering.
  - **Impact**: It selects the fixed price limit, though arbitrary target code may ignore it.

Branches and code coverage

Intended branches

- Zero `callbackAuth` permits entry and commits `(pool, tokenIn, amountIn)`.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`, `PancakeV3PluginTest.test_quote_isFundFreeAndLeavesNoTrace`)
- With `zeroForOne == true`, the live quote selects the lower square-root-price limit.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`)
- With `zeroForOne == false`, quoting selects the upper limit.
  - ☐ Test coverage
- The tested positive `amountIn` casts to the signed exact-input amount and yields a quote.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`)
- Zero and `type(int256).max` boundaries remain untested.
  - ☐ Test coverage
- Normal pool completion ignores declared returns, yields zero, and clears transient state.
  - ☐ Test coverage
- An authenticated positive-delta callback reverts with a raw 32-byte output.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`)
- A propagated callback revert rolls back the pool frame but is caught by the plug-in.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`)
- The exact 32-byte quote payload from a tested live callback is decoded as the returned output amount.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`, `PancakeV3PluginTest.test_quote_throughRouterAmountOutV2`)
- Arbitrary unrestricted 32-byte payload values remain untested.
  - ☐ Test coverage
- A positive `amount0Delta` selects `-amount1Delta` as the quoted output.

- ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`)
- Otherwise, the callback selects `-amount0Delta`.
  - ☐ Test coverage
- Repeated successful quotes leave no token funds or persistent pool-state trace and allow the next quote to proceed.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_isFundFreeAndLeavesNoTrace`)
- Successful completion emits no event and clears every transient field.
  - ☐ Test coverage

Negative behavior

- Nonzero entry value or callback value is rejected by nonpayable dispatch.
  - ☐ Negative test
- Nonzero `callbackAuth` reverts with `PCV3:Reentrancy`.
  - ☐ Negative test
- An `amountIn` above `type(int256).max` reverts before the external call and restores transient state.
  - ☐ Negative test
- Malformed callback data, authentication failure, or nonpositive deltas revert.
  - ☐ Negative test
- Minimum-int negation overflows, while unexpected delta signs may produce zero or nonsensical output.
  - ☐ Negative test
- A non-pool target produces non-quote revert data that is propagated as a failure.
  - ☑ Negative test (`PancakeV3PluginTest.test_RevertWhen_QuoteOnNonPool`)
- Exact byte-for-byte bubbling and transient-state restoration after this failure are not asserted.
  - ☐ Negative test
- A pool that catches callback failure may preserve its effects and return zero.
  - ☐ Negative test
- Any direct 32-byte pool revert is accepted, while other revert lengths are bubbled.
  - ☐ Negative test
- Armed reentrancy fails, and repeated authenticated quote callbacks always revert before payment.
  - ☐ Negative test
- Pool amounts returned by successful calls are ignored because the `try` has no `returns`

clause.

- ☐ Negative test

- Reported 16/16 plug-in tests do not establish per-branch coverage.
  - ☐ Negative test

Function call analysis

- `IPancakeV3Pool(pool).swap(address(this), zeroForOne, SafeCast.toInt256(amountIn), zeroForOne ? PancakeV3Plugin.MIN_SQRT_RATIO_PLUS_ONE : PancakeV3Plugin.MAX_SQRT_RATIO_MINUS_ONE, abi.encode(tokenIn, amountIn))`
  - The caller-selected target receives zero ETH; declared returns are ignored, while propagated 32-byte reverts determine the quote and caught failures may preserve target-side effects.
- `SafeCast.toInt256(amountIn)`
  - This internal conversion preserves valid values and reverts before `pool.swap` on overflow.
- `this.parseRevertReason(reason)`
  - This generated header is incorrect: the exact source uses internal pure `parseRevertReason(reason)`, which decodes 32 bytes or bubbles the exact data without external calls.
- **Manual call-path verification**: The exact source reaches authenticated `pancakeV3SwapCallback`, whose quoting branch cannot transfer tokens. Generated tracing omits that callback and misresolves the internal parser. ASI also fails to parse four EIP-1153 `transient` declarations, so the Cancun semantics were checked against source. Repeated-callback spending exists only in the sibling nonquoting `swap` path.

Function: `pancakeV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes data)`

Transient state authenticates the caller, token, and amount but not the pool's code. Payment transfers the capped positive delta, while quoting reverts with the opposite delta and transfers nothing.

Inputs

- `amount0Delta`
  - **Control**: The authenticated caller-selected pool supplies this delta.

  - **Constraints**: At least one delta must be positive, and quote negation of `type(int256).min` panics, while other invalid signs may wrap.
  - **Impact**: Its sign selects the payment or quote direction and may determine the transferred or returned amount.
- `amount1Delta`
  - **Control**: The authenticated caller-selected pool supplies this delta.
  - **Constraints**: At least one delta must be positive, and payment selects this value only when `amount0Delta <= 0`.
  - **Impact**: It may determine the capped payment or the raw quote payload.
- `data`
  - **Control**: The callback caller supplies bytes decoded as `(tokenIn, amountIn)`, with trailing bytes ignored.
  - **Constraints**: Decoding and commitment authentication must succeed, while initiator `SafeCast` limits reachable `amountIn` to `int256.max`.
  - **Impact**: The decoded token becomes the transfer target, and `amountIn` caps each callback payment.

Branches and code coverage

Intended branches

- Valid ABI data decodes the tested `tokenIn` and `amountIn`.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`, `PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`)
- Ignored trailing callback bytes remain untested.
  - ☐ Test coverage
- The callback proceeds when `msg.sender`, `tokenIn`, and `amountIn` match the armed transient commitment.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`, `PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`)
- Processing continues for both tested standard sign patterns, each with one positive and one negative delta.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`, `PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`, `PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- The nonstandard both-positive case remains untested.
  - ☐ Test coverage
- In quote mode, the authenticated callback returns the tested output through an exact 32-byte revert while requiring no token funds.

- ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`, `PancakeV3PluginTest.test_quote_isFundFreeAndLeavesNoTrace`)

- In the tested quote direction, the callback uses the negated opposite delta.
  - ☑ Test coverage (`PancakeV3PluginTest.test_quote_matchesSwap`)
- Other sign patterns can yield zero, wrapped, or minimum-integer panic behavior.
  - ☐ Test coverage
- Payment direction selects positive `amount0Delta`, otherwise positive `amount1Delta`, as `owed`.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`, `PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- Payment proceeds when `owed <= amountIn`, with equality permitted.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`, `PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- Repeated callback spending: each authenticated callback can spend up to `amountIn`; a dedicated 1/1 reproduction showed cumulative payment above that amount when additional balance exists.
  - ☐ Test coverage

Negative behavior

- Malformed callback data reverts during ABI decoding before authentication or token movement.
  - ☐ Negative test
- An unarmed direct caller or a callback with a different token reverts with `PCV3:Auth`.
  - ☑ Negative test (`PancakeV3PluginTest.test_RevertWhen_CallbackCalledDirectly`, `PancakeV3PluginTest.test_RevertWhen_CallbackTokenMismatch`)
- A separately altered encoded amount remains untested.
  - ☐ Negative test
- Two nonpositive deltas revert with `"PCV3:Delta"`.
  - ☐ Negative test
- Negating `type(int256).min` in quote mode panics instead of producing the intended payload.
  - ☐ Negative test
- A payment request with `owed > amountIn` reverts with `"PCV3:Overpay"`.
  - ☑ Negative test (`PancakeV3PluginTest.test_RevertWhen_PoolAsksForMoreThanAmountIn`)
- A failed, false, too-short, or malformed token response reverts and rolls back

propagated effects.

- ☐ Negative test

- Nonzero native value is rejected by the nonpayable dispatcher before the function body executes.
  - ☐ Negative test

Function call analysis

- `TransferHelper.safeTransfer(tokenIn, msg.sender, owed)`
  - This in-lined helper makes the sole external edge, a low-level `tokenIn.transfer(msg.sender, owed)` call that accepts empty or true returns — including no-code targets — without verifying balances or token semantics.
- **Manual call-path verification**: Authorization remains armed during the call, so initiator reentry fails while the committed pool may repeat or catch callbacks. ASI showed the wrapper because parsing rejected four EIP-1153 `transient` declarations; source review confirms the nested token call and atomic rollback behavior.

Function: `swap(address pool, address tokenIn, uint256 amountIn, bool zeroForOne, address recipient)`

The permissionless `swap` function authenticates callbacks for the active call but trusts arguments selected by the caller and deltas reported by the pool. Reported spend determines refunds, but output to `recipient` is not verified. The executor later checks the actual final `tokenOut` balance against `amountOutMinimum`, although ambient balance can satisfy that check.

Inputs

- `pool`
  - **Control**: The direct caller or user-selected route controls the pool address.
  - **Constraints**: No factory, code, pair, or allowlist validation is performed.
  - **Impact**: The pool controls callbacks, returned deltas, and actual output delivery.
- `tokenIn`
  - **Control**: The direct caller or route supplies the token independently of the pool.
  - **Constraints**: It is committed for callbacks but not validated against the pool pair or token code.
  - **Impact**: Its code controls callback payments, refunds, and token-driven reentrancy.

- `amountIn`
  - **Control**: The caller supplies it. During routed execution, the executor replaces it with the step amount.
  - **Constraints**: Values from zero through `type(int256).max` pass the cast. Each callback payment is capped at `amountIn`.
  - **Impact**: It sets the pool amount, callback cap, and refund baseline without proving prefunding.
- `zeroForOne`
  - **Control**: The caller or route controls this direction flag.
  - **Constraints**: It selects the price limit and returned-delta interpretation without validating token ordering.
  - **Impact**: Noncanonical signed deltas can suppress refunds, inflate output, or trigger negation panic.
- `recipient`
  - **Control**: The caller freely selects the pool's output recipient.
  - **Constraints**: It is neither authenticated nor included in plug-in–side output accounting.
  - **Impact**: The plug-in recipient can divert or fabricate output; top-level router-recipient trapping is a separate path.

Branches and code coverage

Intended branches

- Exact EIP-1153 semantics make `callbackAuth` and `quoting` transaction-scoped transient state.
  - ☐ Test coverage
- An idle `callbackAuth` permits committing `(pool, tokenIn, amountIn)` and starting the operation.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`, `PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- The tested positive input amounts cast successfully to signed exact-input amounts.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`, `PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- Zero and `type(int256).max` boundaries remain untested.
  - ☐ Test coverage
- With `zeroForOne == true`, the live pool consumes token0-side input and produces asserted positive output.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`)

- The exact lower price-limit constant is not asserted.
  - ☐ Test coverage
- With `zeroForOne == false`, the live pool consumes token1-side input and produces asserted positive output.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- The exact upper price-limit constant is not asserted.
  - ☐ Test coverage
- An authenticated positive callback pays the selected delta, individually capped by `amountIn`.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`, `PancakeV3PluginTest.test_swap_WETH_USDC_Pancake`)
- Pool returns are ABI-decoded, and sequential swaps prove the transient authorization is cleared between operations.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_SplitOverTwoPancakePools`)
- When the mock pool spends 60% of input, the plug-in refunds the exact 40% remainder to its caller and retains no input.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_PartialFillRefundsTheRemainder`)
- Residue in the router executor is not exercised by this direct plug-in test.
  - ☐ Test coverage
- A full-input live swap completes without an input refund and leaves no input in the plug-in.
  - ☑ Test coverage (`PancakeV3PluginTest.test_swap_USDC_WETH_Pancake`)
- A reported spend above `amountIn` skips refund without rejection.
  - ☐ Test coverage
- Supplied suite totals do not establish exact per-branch coverage.
  - ☐ Test coverage

Negative behavior

- Nonzero value and malformed calldata are rejected before the body; `tx.origin` is unused.
  - ☐ Negative test
- A nonzero in-flight `callbackAuth` rejects a nested `swap` with `PCV3:Reentrancy`.
  - ☑ Negative test (`PancakeV3PluginTest.test_RevertWhen_PoolReenters`)
- Nested `getAmountOut` is not separately exercised.

☐ Negative test

- An `amountIn` above `type(int256).max` reverts and restores the transient commitment.

☐ Negative test

- An unarmed direct callback and a callback with a mismatched token revert.

☑ Negative test (`PancakeV3PluginTest.test_RevertWhen_CallbackCalledDirectly`, `PancakeV3PluginTest.test_RevertWhen_CallbackTokenMismatch`)

- Malformed callback ABI and a separately mismatched amount remain untested.

☐ Negative test

- A selected callback debt above `amountIn` reverts with `PCV3:Overpay`.

☑ Negative test (`PancakeV3PluginTest.test_RevertWhen_PoolAsksForMoreThanAmountIn`)

- Nonpositive deltas and malformed callback conditions remain untested.

☐ Negative test

- Repeated authenticated callbacks remain individually capped but can cumulatively exceed `amountIn`.

☐ Negative test

- Propagated pool, cast, returndata, arithmetic, or token failures atomically revert the call.

☐ Negative test

- The mock pool fabricates deltas and delivers no output token, yet the plug-in reports a positive `amountOut`.

☑ Negative test (`PancakeV3PluginTest.test_swap_PartialFillRefundsTheRemainder`)

- A pool may also omit callbacks or catch callback failures.

☐ Negative test

- The tested refund token returns `true`, and the unspent input is returned successfully.

☑ Negative test (`PancakeV3PluginTest.test_swap_PartialFillRefundsTheRemainder`)

- Empty return data is also accepted; failure, false, or malformed data is rejected.

☐ Negative test

- Refund token code may begin a fresh operation after `callbackAuth` is cleared.

☐ Negative test

- Supplied suite and proof descriptions do not establish exact negative-test mappings.

☐ Negative test

Function call analysis

- `IPancakeV3Pool(pool).swap(recipient, zeroForOne, SafeCast.toInt256(amountIn), zeroForOne ? PancakeV3Plugin.MIN_SQRT_RATIO_PLUS_ONE : PancakeV3Plugin.MAX_SQRT_RATIO_MINUS_ONE, abi.encode(tokenIn, amountIn))`
    - The untrusted pool may invoke the callback repeatedly, catch failures, fabricate signed deltas, or omit output to `recipient`; any propagated failure atomically reverts the call.
- `SafeCast.toInt256(amountIn)`
    - This deterministic internal cast accepts values from zero through `int256.max`; larger values revert before the pool call.
- `TransferHelper.safeTransfer(tokenIn, msg.sender, amountIn - spent)`
    - The untrusted token may reenter or report success without performing the expected transfer; failures revert, while empty returndata from a successful call is accepted.
- **Manual call-path verification**: On the pool's omitted-callback path, `tokenIn` is paid to the authenticated pool. Exact ASI parsing failed on four EIP-1153 `transient` declarations, leaving the resulting traces incomplete. Source-unqualified constants were rendered separately with qualification.

5.23. Module: PauseLib.sol

Function: `pause()`

The `PauseLib.pause()` function sets the host/proxy pause flag and emits `PauseModule.Paused()`, reverting when already paused. It takes no parameters and returns no values. Only the owner-gated wrapper supplies authorization.

Branches and code coverage

Intended branches

- If the fixed-slot `isPaused` value is false, the owner-gated route writes true.
    - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Pause_succeeds`)
- Success emits `PauseModule.Paused()` from the host.
    - ☐ Test coverage
- Pausing an already paused host reverts with `IsPaused()`.
    - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Pause_IsPaused_reverts`)
- The test does not separately assert unchanged storage or absence of logs.

- ☐ Test coverage

- Because none of the supplied descendants overrides `PauseModule._pause()`, `FrxUSDOFTUpgradeable.pause()` resolves through `_pause()` to this linked `PauseLib.pause()` declaration.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Pause_succeeds`)
- The nonpayable wrapper rejects nonzero `msg.value` before delegation, although the generated `DELEGATECALL` would otherwise preserve the outer value.
  - ☐ Test coverage

Negative behavior

- A non-owner caller is rejected by the host wrapper.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Pause_OnlyOwner`)
- Reversion before linked-library dispatch is not instrumented.
  - ☐ Negative test
- An already paused state propagates `IsPaused()` through the wrapper.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Pause_IsPaused_reverts`)
- Rollback of state and logs is not independently asserted.
  - ☐ Negative test
- Nonzero native value reverts at the wrapper before target execution, leaving pause storage unchanged and emitting no event.
  - ☐ Negative test

Function call analysis

- `this._getPauseStorage()`
  - ASI misrenders this as an external call. The source calls a private pure `_getPauseStorage()` helper that selects constant slot `0x242e96562e738afaa26e2a4eb34bb6ab09c26cd11226ccf8fbc02da32756d300`, matching `PauseModule.PauseStorageLocation`, without external control transfer.
- **Manual call-path verification**: The `FrxUSDOFTUpgradeable.pause()` function applies `onlyOwner` and rejects value before `_pause()` invokes the linked library by `DELEGATECALL`. The delegatecall preserves the sender, value, and host storage context while propagating failures. The target performs only the fixed-slot check, write, and event. A direct state-changing `CALL` to the library bytecode is compiler rejected. Unrelated ASI parser failures are confined to Tempo `_quote` paths.

Function: unpause()

The `PauseLib.unpause()` function requires the fixed namespaced pause flag to be true, clears it, and emits `PauseModule.Unpaused()` in the consumer's context under linked `DELEGATECALL`. The owner-gated, nonpayable FrxUSDOFTUpgradeable wrapper enforces authorization and zero value. Correct linked bytecode is assumed.

Branches and code coverage

Intended branches

- When `isPaused == true`, the function writes `false` in the consumer context.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Unpause_succeeds`)
- Success emits `PauseModule.Unpaused()` from the consumer context.
  - ☐ Test coverage
- The `_getPauseStorage()` helper binds only to constant slot `0x242e96562e738afaa26e2a4eb34bb6ab09c26cd11226ccf8fbc02da32756d300`, matching `PauseModule`.
  - ☐ Test coverage
- The supplied route is owner-gated `FrxUSDOFTUpgradeable.unpause()` → inherited `PauseModule._unpause()` → linked `PauseLib.unpause()`.
  - ☑ Test coverage (`FrxUSDOFTUpgradeableTest.test_Unpause_succeeds`)
- Solc 0.8.22 uses `DELEGATECALL`, preserving outer `msg.value`; the harness passed 2/2 linked calls with `msg.value == 1`, while the supplied wrapper admits only zero value.
  - ☐ Test coverage

Negative behavior

- Unpausing an already unpaused host reverts with `NotPaused()`.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Unpause_NotPaused_reverts`)
- The test does not separately assert unchanged storage or absence of logs.
  - ☐ Negative test
- A non-owner caller is rejected by the host wrapper.
  - ☑ Negative test (`FrxUSDOFTUpgradeableTest.test_Unpause_OnlyOwner`)
- Reversion before linked-library dispatch is not instrumented.
  - ☐ Negative test
- Nonzero `msg.value` reverts at the nonpayable wrapper before delegation, although the library body itself does not inspect value.

- ☐ Negative test

- A normal `CALL` to the deployed state-changing library is rejected by Solidity's library-runtime delegatecall guard.
  - ☐ Negative test
- Any linked-call failure propagates atomically, leaving storage and logs unchanged.
  - ☐ Negative test

Function call analysis

- `this._getPauseStorage()`
  - This ASI trace is misresolved because `_getPauseStorage()` is a private pure helper that only assigns the fixed storage slot and performs no external call.
- `PauseModule._unpause()` → `PauseLib.unpause()` (manual interaction omitted by the generated FCA).
  - The actual edge is a linked `DELEGATECALL` with no arguments or returns; it preserves consumer storage, sender, and value and propagates failure, and the reviewed body exposes no callback or reentrancy opportunity.
- **Manual call-path verification**: Source review confirms the owner-gated, nonpayable wrapper route, the fixed linked target assumption, and that ASI omitted the actual module-to-library `DELEGATECALL`.

5.24. Module: PermitLib.sol

Function: `validatePermit(address owner, address spender, uint256 value, uint256 deadline, bytes signature)`

The linked PermitLib library verifies a deadline-bound, nonce-consuming, proxy-domain EOA/ERC-1271 permit before PermitModule approves. Validation emits nothing and moves no assets.

Inputs

- `owner`
  - **Control**: A relayer supplies the owner, whose EOA signature or ERC-1271 policy must authorize the permit.
  - **Constraints**: The owner must be nonzero, pass signature validation, and have a nonce below `type(uint256).max`.
  - **Impact**: It selects the nonce, signed owner, allowance owner, and possible ERC-1271 target.
- `spender`

  - **Control**: A relayer supplies the signature-bound spender.
  - **Constraints**: Validation permits any address, but the enclosing `_approve` rejects zero.
  - **Impact**: It selects the allowance key and emitted `Approval` spender.
- `value`
  - **Control**: A relayer supplies the signature-bound allowance amount.
  - **Constraints**: Any `uint256` is accepted unchanged.
  - **Impact**: It becomes the exact approved allowance without moving assets.
- `deadline`
  - **Control**: A relayer supplies the signature-bound deadline.
  - **Constraints**: It must satisfy `block.timestamp <= deadline`.
  - **Impact**: Expiry reverts before nonce access or signature interaction.
- `signature`
  - **Control**: A relayer supplies arbitrary bytes, or the alternate overload packs `r || s || v`.
  - **Constraints**: It must authorize the derived EIP-712 digest through ECDSA or ERC-1271.
  - **Impact**: It proves authorization and may be passed to a value-free `STATICCALL` without being stored.

Branches and code coverage

Intended branches

- Either nonpayable permit overload reaches the fixed linked library by generated `DELEGATECALL`, with `v/r/s` packed when applicable.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- A deadline later than the current timestamp continues.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- A deadline equal to the current timestamp continues.
  - ☐ Test coverage
- The `_useNonce(owner)` call hashes the preincrement nonce and stores its checked increment.
  - ☐ Test coverage
- The name hash selects the stored name, legacy hash, or the empty-string hash.
  - ☐ Test coverage
- The version hash selects the stored version, legacy hash, or the empty-string hash.
  - ☐ Test coverage

- The domain binds name, version, chain ID, and proxy-context `address(this)`.
  - ☐ Test coverage
- A matching ECDSA recovery accepts without invoking ERC-1271.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- Failed ECDSA recovery triggers an ERC-1271 `STATICCALL`, whose magic value accepts.
  - ☐ Test coverage
- Valid authorization writes the exact tested allowance.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- Valid authorization increments the nonce and emits `Approval`.
  - ☐ Test coverage
- Zero and maximum allowance values can succeed.
  - ☐ Test coverage
- FrxUSD pause and freeze hooks do not affect approval validation.
  - ☐ Test coverage

Negative behavior

- Nonzero value is rejected by scoped nonpayable wrappers, although generated library delegation itself preserves value.
  - ☐ Negative test
- Solidity's library guard rejects an ordinary state-changing direct `CALL`.
  - ☐ Negative test
- An expired deadline reverts before nonce consumption or signature interaction.
  - ☐ Negative test
- A maximum nonce overflows on increment and reverts atomically.
  - ☐ Negative test
- Malformed, mismatched, or incorrect-domain signatures revert with `SignatureModule.InvalidSignature()`.
  - ☐ Negative test
- Failed, malformed, short, or nonmagic ERC-1271 responses invalidate the signature and restore the nonce.
  - ☐ Negative test
- A zero owner cannot pass the explicit signer check.
  - ☐ Negative test
- A zero spender may validate, but `_approve` later reverts and restores the nonce.

- ☐ Negative test

- Misconfigured name or version state causes domain mismatch rather than caller-controlled substitution.
  - ☐ Negative test
- Library or approval failure bubbles and atomically removes the nonce update and any log.
  - ☐ Negative test

Function call analysis

- `this._requireIsValidSignatureNow(signer: owner, structHash: keccak256(abi.encode(PermitLib.PERMIT_TYPEHASH, owner, spender, value, this._useNonce(owner), deadline)), signature: signature) -> SignatureChecker.isValidSignatureNow(signer: signer, hash: ECDSAUpgradeable.toTypedDataHash(this._domainSeparatorV4(), structHash), signature: signature)`
  - ECDSA failure conditionally invokes value-free ERC-1271 `STATICCALL`; only a successful magic response accepts, and static context prevents mutation.
- `this._requireIsValidSignatureNow(signer: owner, structHash: keccak256(abi.encode(PermitLib.PERMIT_TYPEHASH, owner, spender, value, this._useNonce(owner), deadline)), signature: signature) -> ECDSAUpgradeable.toTypedDataHash(this._domainSeparatorV4(), structHash)`
  - This internal pure helper derives the digest from signed fields, stored nonce, configured domain, chain ID, and proxy address.
- `this._requireIsValidSignatureNow(signer: owner, structHash: keccak256(abi.encode(PermitLib.PERMIT_TYPEHASH, owner, spender, value, this._useNonce(owner), deadline)), signature: signature) -> this._domainSeparatorV4() -> this._EIP712NameHash() -> this._getEIP712Storage()`
  - Manual review confirms the Yul only assigns the fixed EIP-712 storage slot and makes no external call.
- `this._requireIsValidSignatureNow(signer: owner, structHash: keccak256(abi.encode(PermitLib.PERMIT_TYPEHASH, owner, spender, value, this._useNonce(owner), deadline)), signature: signature) -> this._domainSeparatorV4() -> this._EIP712VersionHash() -> this._getEIP712Storage()`
  - Manual review confirms the version path only reads and hashes fixed-slot state without external interaction.
- `this._useNonce(owner) -> this._getPermitModuleStorage()`
  - The Yul assigns the constant permit-storage slot, after which checked nonce

increment either succeeds or reverts.

- **Manual call-path verification**: The real path is wrapper → fixed-library `DELEGATECALL` → nonce/domain/signature logic → optional ERC-1271 `STATICCALL` → `_approve`. Generated FCA mislabels internals and omits these external edges. Solc 0.8.22 confirms that the delegatecall preserves value while the wrappers enforce zero. EIP-712 setup remains limited to initializer/reinitializer execution. ASI failures at `contracts/FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `contracts/FraxOFTUpgradeableTempo.sol@48:2180` do not truncate this path.

5.25. Module: PermitModule.sol

Function: `permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, byte[32] r, byte[32] s)`

Permissionless relayers submit EIP-712 permits; successful validation consumes the owner's nonce, sets allowance exactly to `value`, and emits `Approval`. The preserved signature uses `byte[32]`, while the Solidity entry point uses `bytes32`; the function returns nothing and transfers no tokens or native value.

Inputs

- `owner`
  - **Control**: Any relayer may supply `owner`, but success requires that owner's valid EOA or ERC-1271 authorization.
  - **Constraints**: The signature binds `owner`, and a zero address or invalid authorization reverts with all tentative changes rolled back.
  - **Impact**: It selects the nonce, signer or callback target, allowance owner, and `Approval` owner.
- `spender`
  - **Control**: The relayer supplies `spender`, but the owner's signature binds it.
  - **Constraints**: Concrete `_approve` implementations reject the zero address and roll back the consumed nonce.
  - **Impact**: It selects the allowance recipient and `Approval` spender without receiving tokens or native value.
- `value`
  - **Control**: The relayer supplies the full `uint256`, which is bound by the signature.
  - **Constraints**: It is neither narrowed nor capped and passes unchanged to `_approve`.
  - **Impact**: It replaces the existing allowance exactly and appears in `Approval`.
- `deadline`

  - **Control**: The relayer supplies `deadline`, which is bound by the signature.
  - **Constraints**: Validation accepts `block.timestamp <= deadline`, including equality.
  - **Impact**: It determines the permit's final valid timestamp without being stored.
- `v`
  - **Control**: The relayer supplies this signature byte.
  - **Constraints**: It is packed after `r` and `s`, with ECDSA or ERC-1271 determining validity.
  - **Impact**: It affects signature acceptance but is not independently stored or emitted.
- `r`
  - **Control**: The relayer supplies this signature component.
  - **Constraints**: The Solidity type is `bytes32`, occupying bytes 0–31 of `r || s || v`.
  - **Impact**: It affects signature acceptance and whether nonce and allowance changes commit.
- `s`
  - **Control**: The relayer supplies this signature component.
  - **Constraints**: The Solidity type is `bytes32`, occupying bytes 32–63 and subject to ECDSA low-`s` rules for EOAs.
  - **Impact**: It affects signature acceptance and whether nonce and allowance changes commit.
- Implicit inputs are `block.timestamp`, `block.chainid`, `address(this)`, the stored EIP-712 name/version, `nonces[owner]`, and linked-library bytecode; neither `msg.sender` nor `tx.origin` enters the digest.

Branches and code coverage

Intended branches

- Wrapper packing and internal dispatch: canonical calldata with zero value packs `r || s || v` and internally invokes the bytes-signature overload.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- Deadline boundary succeeds: validation accepts `block.timestamp == deadline` and then consumes the current nonce.
  - ☐ Test coverage
- EOA signature succeeds: valid ECDSA recovery authorizes approval for a noncontract owner.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)

- ERC-1271 signature succeeds: a contract owner succeeds only when its `STATICCALL` returns the ERC-1271 magic value.
  - ☐ Test coverage
- Concrete approval writes exactly `value` to the tested allowance.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- Concrete approval emits `Approval` without invoking pause or freeze hooks.
  - ☐ Test coverage

Negative behavior

- Nonzero native value is rejected before entry: the nonpayable dispatcher rejects `msg.value != 0`.
  - ☐ Negative test
- Malformed or noncanonical calldata is rejected during ABI decoding: invalid lengths or encodings prevent function-body execution.
  - ☐ Negative test
- Expired permit is rejected: a call where `block.timestamp > deadline` reverts before nonce consumption or signature checking.
  - ☐ Negative test
- Invalid EOA signature is rejected: invalid recovery data, noncanonical `s`, unsupported `v`, or signer mismatch reverts and rolls back the nonce.
  - ☐ Negative test
- Failed ERC-1271 authorization is rejected: callback failure, malformed output, or incorrect magic value yields `InvalidSignature`.
  - ☐ Negative test
- Zero owner is rejected: the PermitLib library rejects `address(0)` and rolls back its tentative nonce increment.
  - ☐ Negative test
- Zero spender is rejected by concrete approval: the `_approve` function reverts and atomically restores the nonce.
  - ☐ Negative test
- Replay is rejected: a successful permit increments the nonce, making the prior signature invalid.
  - ☐ Negative test
- Nonce overflow is rejected: checked nonce incrementation panics at `type(uint256).max`.
  - ☐ Negative test
- Approval failure rolls back validation effects: any `_approve` revert rolls back nonce,

allowance, nested effects, and logs.

- ☐ Negative test

Function call analysis

- `this.permit(owner: owner, spender: spender, value: value, deadline: deadline, signature: abi.encodePacked(r, s, v)) -> PermitLib.validatePermit(owner, spender, value, deadline, signature)`
  - The rendered `this.permit` is an internal overload call followed by a fixed-target linked-library `DELEGATECALL` in token or proxy storage context.
- `owner.isValidSignature(digest, signature)`
  - Contract owners receive a read-only ERC-1271 `STATICCALL` after tentative nonce increment, and failure or incorrect magic causes `InvalidSignature`.
- **Manual call-path verification**: Execution is nonpayable dispatch → signature packing → internal overload → PermitLib delegatecall → EOA recovery or conditional ERC-1271 staticcall → internal `_approve`, with atomic rollback on every revert.

Function: `permit(address owner, address spender, uint256 value, uint256 deadline, bytes signature)`

Any relayer may submit an ERC-2612–style signature, while validation binds authorization to the owner, nonce, deadline, chain, proxy, and EIP-712 domain. Successful concrete implementations replace the allowance through `ERC20Upgradeable._approve`, emit `Approval`, transfer no value, and return no data.

Inputs

- `owner`
  - **Control**: The caller selects `owner`, but only its valid EOA signature or ERC-1271 response authorizes the permit.
  - **Constraints**: Validation requires a nonzero owner, valid domain-bound authorization, and a checked increment of `nonces[owner]`.
  - **Impact**: It selects the signer, nonce namespace, allowance owner, and possible ERC-1271 callback target.
- `spender`
  - **Control**: The caller selects `spender`, which is cryptographically bound into the signed permit.
  - **Constraints**: The `ERC20Upgradeable._approve` function rejects the zero address after signature validation.
  - **Impact**: It selects the allowance beneficiary and indexed spender in

`Approval`.

- `value`
  - **Control**: The caller supplies the full `uint256`, subject to the owner's authorization.
  - **Constraints**: It is neither bounded nor transformed before hashing and approval.
  - **Impact**: It replaces the `(owner, spender)` allowance without transferring tokens.
- `deadline`
  - **Control**: The caller supplies the signed expiration timestamp.
  - **Constraints**: Validation requires `block.timestamp <= deadline`, including equality.
  - **Impact**: It determines the permit's final valid timestamp.
- `signature`
  - **Control**: The caller supplies arbitrary bytes, while the alternate overload packs `r || s || v`.
  - **Constraints**: The SignatureChecker library requires valid EOA recovery or an ERC-1271 magic response for the exact digest.
  - **Impact**: It authorizes the signed tuple and may trigger a no-value `STATICCALL` to a contract owner.

Branches and code coverage

Intended branches

- A zero-value nonpayable call proceeds regardless of relayer identity.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- A deadline later than `block.timestamp` is accepted.
  - ☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)
- A deadline equal to `block.timestamp` is accepted.
  - ☐ Test coverage
- The current owner nonce is included in the digest and incremented once with checked arithmetic.
  - ☐ Test coverage
- The digest binds all permit fields to the stored EIP-712 name/version, current chain ID, and proxy address.
  - ☐ Test coverage
- A valid EOA signature authorizes the permit without an external callback.

☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)

- A contract owner may authorize through a successful ERC-1271 `STATICCALL` returning the magic value.

☐ Test coverage

- Concrete dispatch replaces the tested ERC-20 allowance.

☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)

- Concrete dispatch emits `Approval` and returns no data.

☐ Test coverage

- The alternate overload packs `r`, `s`, and `v` and internally enters this five-argument path.

☑ Test coverage (`FraxOFTUpgradeableTest.test_Permit_succeeds`)

Negative behavior

- Nonzero `msg.value` is rejected before the function body executes.

☐ Negative test

- An expired deadline reverts before nonce access, signature checking, or approval.

☐ Negative test

- A maximum nonce causes checked-addition panic and leaves state unchanged.

☐ Negative test

- Any malformed, mismatched, stale, or incorrect-domain EOA signature reverts with `InvalidSignature`.

☐ Negative test

- A zero owner is rejected with `InvalidSignature`.

☐ Negative test

- A failed, reverting, malformed, short, or nonmagic ERC-1271 response invalidates authorization.

☐ Negative test

- A zero spender is rejected by `_approve`, rolling back the nonce increment.

☐ Negative test

- Any other approval revert atomically rolls back nonce, allowance, and logs.

☐ Negative test

Function call analysis

- `PermitLib.validatePermit(owner, spender, value, deadline, signature)`

- The deployment-fixed library executes via a no-value `DELEGATECALL` using the proxy's storage and context. The SignatureChecker library may make a no-value, owner-selected ERC-1271 `STATICCALL`; a failed or nonmagic response causes validation to revert.

- **Manual call-path verification**: Generated tracing omits validation internals and `_approve`. Source review confirms domain-bound nonce validation followed by concrete `ERC20Upgradeable._approve`. ASI builder failures for FraxOFTMintableAdapterUpgradeableTIP20.sol and FraxOFTUpgradeableTempo.sol do not change the conclusion that only Tempo exposes and inherits this unchanged path.

5.26. Module: RateLimiterLib.sol

Function: `checkpointRateLimits(uint32[] _eids)`

Through a linked `DELEGATECALL`, the function checkpoints each supplied EID in host storage, writes the derived states, and emits `RateLimitCheckpointed`. Empty arrays do nothing.

Inputs

- `_eids`
  - **Control**: The caller controls array length, order, duplicates, and EID keys, while this declaration performs no authorization or value checks.
  - **Constraints**: ABI-valid `uint32[]` values are accepted without uniqueness, ordering, enablement, configuration, or length validation beyond gas limits.
  - **Impact**: Stored configurations, states, and `uint32(block.timestamp)` determine each write and event; the function does not read the preserved caller context or value.

Branches and code coverage

Intended branches

- An empty `_eids` array skips all configuration reads, state writes, and events.
  - ☐ Test coverage
- Each iteration checkpoints `_eids[i]`, increments `i`, and exits when `i == _eids.length`.
  - ☐ Test coverage
- The `overrideDefaultConfig` field selects either the complete per-EID configuration or the default configuration.
  - ☐ Test coverage

- If `lastUpdated == 0` or is at least the truncated current timestamp, usage is preserved and the timestamp is updated.
  - ☐ Test coverage
- If time advanced, outbound and inbound usage decay independently before the shared timestamp advances.
  - ☐ Test coverage
- Zero usage, limit, window, or elapsed time preserves that direction's usage, including when a disabled direction uses a zero-valued placeholder configuration.
  - ☐ Test coverage
- A full elapsed window clears usage; otherwise, floored replenishment is either zero or is subtracted from usage, discarding fractional recovery below one usage unit.
  - ☐ Test coverage
- Every processed element writes the complete state and emits `RateLimitCheckpointed`, regardless of enablement flags.
  - ☐ Test coverage
- Successful execution returns no value, while any revert or gas exhaustion rolls back the entire operation.
  - ☐ Test coverage

Negative behavior

- This library declaration has no owner or service-admin check, although supplied host wrappers are independently `onlyOwner`.
  - ☐ Negative test
- Malformed ABI input reverts, preserved nonzero delegatecall value is ignored, and direct state-changing `CALL` is blocked by the library guard.
  - ☐ Negative test
- Zero, unknown, repeated, and unordered EIDs are accepted, with duplicates producing repeated writes and logs.
  - ☐ Negative test
- A sufficiently large array can exhaust gas, atomically reverting all prior writes and logs.
  - ☐ Negative test
- Configuration and state are not revalidated, so type-valid disabled placeholders or usage above the effective limit are processed.
  - ☐ Negative test
- Global or directional disablement does not bypass checkpointing; zero limits or windows for inactive directions preserve historical usage.
  - ☐ Negative test

- After the year-2106 timestamp wrap, a prior high timestamp preserves usage and resets `lastUpdated` to the truncated value.
  - ☐ Negative test
- The reviewed body and internal helpers contain no nested external calls, callbacks, token transfers, or reentrancy points.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimit(_eids[i], this._effectiveRateLimitConfig(_eids[i])) -> this._getRateLimiterStorage()`
  - This is an internal path to a fixed namespaced storage slot, not an external self-call; failure propagates atomically.
- `this._checkpointRateLimit(_eids[i], this._effectiveRateLimitConfig(_eids[i])) -> this._currentRateLimitState(_eid, _config) -> this._getRateLimiterStorage()`
  - This internal path derives state from stored usage, effective configuration, and truncated time without callbacks.
- `this._effectiveRateLimitConfig(_eids[i]) -> this._getRateLimiterStorage()`
  - This internal path selects the per-EID override or default configuration from the fixed host-storage namespace.
- **Manual call-path verification**: The host enters the linked library by `DELEGATECALL`, preserving host storage and context. The trace mislabels internal helpers and omits `_decayUsage` and the delegatecall boundary. Off-path Tempo parser failures do not affect this function.

Function: `consumeRateLimit(uint32 _eid, uint256 _amountLD, bool _isOutbound)`

A fixed-target `DELEGATECALL` tracks directional usage for each EID in caller storage. Disabled paths are no-ops; enabled paths decay usage, revert when capacity is exceeded, or persist state and emit `RateLimitConsumed`.

Inputs

- `_eid`
  - **Control**: Outbound wrappers pass the requested destination EID, while authenticated inbound wrappers pass the source EID.

    - **Constraints**: ABI encoding limits it to `uint32`, but this function performs no peer, support, or nonzero validation.
    - **Impact**: It selects the effective configuration and state bucket; events and errors report it unchanged.
- `_amountLD`
    - **Control**: Outbound paths pass the dust-adjusted debit amount, while inbound paths pass the received local-decimal amount.
    - **Constraints**: Zero returns immediately, and enabled accounting requires the amount not to exceed available capacity.
    - **Impact**: It increments the selected usage and is reported unchanged in `RateLimitConsumed` or `RateLimitExceeded`.
- `_isOutbound`
    - **Control**: Scoped outbound and inbound wrappers fix this value to `true` and `false`, respectively.
    - **Constraints**: The selected direction must be enabled to reach accounting; otherwise, the function returns before accounting.
    - **Impact**: It selects the limit, usage, enable flag, updated state member, and reported direction.

Branches and code coverage

Intended branches

- A zero `_amountLD` returns before storage access or event emission, including with preserved nonzero `msg.value`.
    - ☐ Test coverage — audit-only probe; no scoped-repository test case was mapped
- Nonzero `msg.value` does not block the linked-library call because this body never reads it.
    - ☐ Test coverage — audit-only probe; no scoped-repository test case was mapped
- A globally disabled limiter returns before per-EID configuration or state checkpointing.
    - ☐ Test coverage
- For tested EIDs without overrides, `_effectiveRateLimitConfig` selects the current default.
    - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- When enabled, a per-EID override is selected instead of the default.

- ☐ Test coverage

- Enabled outbound accounting uses the configured outbound limit and usage.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterModuleTest.test_OFTOutboundRateLimit_RefillsLinearly`)
- Outbound accounting returns when `config.outboundEnabled` is false.
  - ☐ Test coverage
- Enabled inbound accounting uses the configured inbound limit and usage.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- Inbound accounting returns when `config.inboundEnabled` is false.
  - ☐ Test coverage
- Initial state incurs no decay, while an older timestamp decays the tested usage before consumption.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterModuleTest.test_OFTOutboundRateLimit_RefillsLinearly`, `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- A future `lastUpdated` skips decay.
  - ☐ Test coverage
- The `_decayUsage` function returns unchanged usage when usage, limit, window, or elapsed time is zero.
  - ☐ Test coverage
- A full tested window restores full inbound capacity, while half an outbound window restores half the capacity.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RefillsLinearly`, `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- Near-overflow arithmetic boundaries remain untested.
  - ☐ Test coverage
- Availability is zero when tested usage equals the limit and otherwise equals the asserted remaining capacity.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterModuleTest.test_OFTOutboundRateLimit_RefillsLinearly`, `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)

- A stored usage above the configured limit is not exercised.
  - ☐ Test coverage
- An amount above available outbound or inbound capacity reverts with `RateLimitExceeded`.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- Ordering before storage writes or event emission is not independently asserted.
  - ☐ Test coverage
- A permitted amount increments the selected usage and stores the resulting state, as observed through remaining capacity.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- A permitted consumption emits `RateLimitConsumed`.
  - ☐ Test coverage
- Downstream token, telemetry, compliance, or endpoint failure atomically reverts prior accounting and events.
  - ☐ Test coverage

Negative behavior

- Overcapacity consumption bubbles the exact `RateLimitExceeded(_eid, _isOutbound, _amountLD, available)` payload.
  - ☑ Negative test (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`)
- Postrevert usage and event state are not independently asserted.
  - ☐ Negative test
- Preserved nonzero `msg.value` must not independently cause this library function to revert.
  - ☐ Negative test — audit-only probe — no scoped-repository test case was mapped
- Zero amount, global disablement, or selected-direction disablement must remain successful accounting no-ops.
  - ☐ Negative test

- A tested outbound EID without an override resolves through the default configuration rather than being rejected.
  - ☑ Negative test (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc`
- Per-EID override resolution is not covered here.
  - ☐ Negative test
- No owner or service-admin failure occurs here because authorization is structural and setters are separately gated.
  - ☐ Negative test
- After the `uint32` timestamp wrap, the no-decay branch can apply, requiring migration before year 2106.
  - ☐ Negative test
- Later reverts must undo accounting, while successful reentrancy observes already consumed capacity.
  - ☐ Negative test

Function call analysis

- `this._getRateLimiterStorage()`
  - This source-internal helper only assigns the fixed `RateLimiterStorageLocation` slot and cannot call or reenter.
- `this._effectiveRateLimitConfig(_eid) -> this._getRateLimiterStorage()`
  - This internal path reads the per-EID override or default; replacing the default can redecay existing usage under a new slope.
- `this._currentRateLimitState(_eid, config) -> this._getRateLimiterStorage()`
  - This internal path decays both usages and persists the shared timestamp. Flooring permanently discards any fractional replenishment.
- **Manual call-path verification**: The generated `this.*` traces represent internal calls and omit `_decayUsage` and the fixed-library `DELEGATECALL`. The `DELEGATECALL` preserves caller storage, sender, address, and value. This body initiates no calls. Downstream interactions with the fixed token and endpoint occur after accounting, and their failures revert all effects. Shared per-EID buckets create contention. A downstream unchecked `uint64` conversion can truncate after the full amount has been consumed, and zero-address freezing can block non-owner mint or burn operations. The ASI failures `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` are downstream `_lzSend(...)` calls.

Function: `setDefaultRateLimitConfig(RateLimiterModule.RateLimitConfig _defaultConfig)`

Owner-gated nonpayable wrappers delegatecall a fixed library that validates enabled directions, overwrites the namespaced default, and emits `DefaultRateLimitConfigSet`.

Inputs

- `_defaultConfig`
  - **Control**: The owner selects all seven struct fields, none of which controls the library target or storage namespace.
  - **Constraints**: Each enabled direction requires a nonzero limit and window, while disabled directions and `overrideDefaultConfig` have no additional validation.
  - **Impact**: The struct overwrites two namespaced storage slots, six fields are emitted, and future non-overridden EIDs use the new default.
- `msg.sender` (implicit)
  - **Control**: The external caller determines `msg.sender`, which delegatecall preserves.
  - **Constraints**: Every in-scope concrete wrapper applies `onlyOwner`, while the library body does not read `msg.sender`.
  - **Impact**: The sender controls reachability through authorization but does not affect validation, storage selection, or event data.
- `msg.value` (implicit)
  - **Control**: The caller supplies `msg.value` for the outer call; delegatecall preserves that value but does not transfer a separate value.
  - **Constraints**: Scoped nonpayable wrappers reject nonzero value before authorization, module dispatch, or delegatecall.
  - **Impact**: The scoped wrappers invoke the target with zero value, and the target neither reads nor transfers value.
- `address(this)` and execution storage context (implicit)
  - **Control**: Delegatecall preserves the calling contract or proxy address and storage context.
  - **Constraints**: A compile-time namespace constant fixes the storage pointer, and the module and library constants match.
  - **Impact**: The two-slot write and event occur in the caller or proxy context rather than the standalone library.
- `tx.origin` (implicit)
  - **Control**: The originating transaction determines `tx.origin`, which delegatecall preserves.
  - **Constraints**: Neither the target nor its reviewed wrapper path reads or

validates `tx.origin`.

- **Impact**: It has no effect on authorization, validation, storage, logging, targeting, or value handling.

Branches and code coverage

Intended branches

- The tested OFT and adapter owner-authorized entry points reach this setter.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- Outbound validation accepts tested enabled configurations with nonzero limit/window and disabled configurations with zero fields.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`, `RateLimiterMod-uleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- Inbound validation accepts tested enabled configurations with nonzero limit/window and disabled configurations with zero fields.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_OFTInboundRateLimit_RevertsThenRefills`, `RateLimiterMod-uleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- The tested valid fields become the effective default configuration used by later debit and credit accounting.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-uleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit`)
- Success emits all six projected event fields before the wrapper returns.
  - ☐ Test coverage
- Tested non-overridden EIDs select the newly stored default.
  - ☑ Test coverage (`RateLimiterModuleTest.test_OFTOutboundRateLimit_RevertsWhenCapacityExc` `RateLimiterMod-`

uleTest.test_AdapterRateLimits_ApplyOnDebitAndCredit)

- Existing per-EID overrides remain effective.
  - ☐ Test coverage
- Default-transition path without checkpointing: uncheckpointed usage decays under replacement parameters, unlike usage checkpointed before the change.
  - ☐ Test coverage — a dedicated reproduction in the verified threat model confirms this behavior (1/1): the uncheckpointed case exposed 1,000 units, while the prechange checkpoint case exposed 910 units

Negative behavior

- Unauthorized callers revert at `onlyOwner` before any delegatecall, storage write, or event.
  - ☐ Negative test.
- Enabled outbound limiting with a zero limit or window reverts first with `InvalidRateLimitConfig()`.
  - ☐ Negative test.
- After outbound validation passes, enabled inbound limiting with a zero limit or window reverts with `InvalidRateLimitConfig()`.
  - ☐ Negative test.
- Malformed calldata, noncanonical booleans, and out-of-width integers fail ABI decoding, so no state is persisted.
  - ☐ Negative test.
- Nonzero value is rejected by the outer nonpayable wrapper before its body executes.
  - ☐ Negative test.
- Direct state-changing `CALL` access is blocked by Solidity's library runtime delegatecall guard.
  - ☐ Negative test.
- Disabled-direction placeholders, default `overrideDefaultConfig`, and absent checkpointing are accepted, and usage can persist across disabling and reenabling.
  - ☐ Negative test — the verified threat model reports that a separate 7/7 contract suite matched the zero-valued disable/reenable edge. It also records a removed probe in which usage remained after more than one disabled window. This item is not marked as complete branch-level negative coverage because it combines multiple accepted-input behaviors and the probe was removed.

Function call analysis

- `this._getRateLimiterStorage()`
    - The generated `this.` edge is incorrect; this private Yul helper only binds the fixed namespace slot and makes no external call.
- **Linked-library interaction omitted by the generated trace — `RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`**: Solidity uses a fixed, deployment-linked `DELEGATECALL` that preserves the caller, value, address, and storage context. Validation failures propagate atomically, and the delegated body makes no further external call.
- **Manual call-path verification**: The nonpayable `onlyOwner` wrapper calls the module, which performs a fixed library delegatecall. The library then validates the outbound and inbound settings, overwrites the fixed slot, and emits the event. Both Solc 0.8.22 harness calls (2/2) confirm that there is no independent delegatecall value guard. ASI's Yul edge and the `_quote` builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` are unrelated to this setter.

Function: `setRateLimitConfigs(RateLimiterModule.SetRateLimitConfigParam[] _params)`

This library function checkpoints each EID under its old effective configuration, stores its replacement, and emits ordered checkpoint/configuration logs. The library lacks authorization and returns nothing, while all supplied wrappers enforce `onlyOwner`.

Inputs

- `_params`
    - **Control**: The owner controls array order, duplicate EIDs, and every replacement configuration field through the supplied wrapper.
    - **Constraints**: Enabled directions require nonzero limits and windows, while disabled fields may be zero and batch size is gas-bounded.
    - **Impact**: Each element checkpoints existing usage before replacement, controls future per-EID capacity, and reverts the whole batch on failure.

Branches and code coverage

Intended branches

- An authorized owner passes the supplied wrappers' `onlyOwner` check and reaches the library through the module wrapper.
    - ☐ Test coverage

- An empty array skips the loop and completes without storage writes, logs, or return data.

  ☐ Test coverage
- Each element is processed in order, and duplicate EIDs sequentially observe earlier checkpoint and configuration changes.

  ☐ Test coverage
- Checkpointing uses the stored per-EID configuration when `overrideDefaultConfig` is true and the default configuration otherwise.

  ☐ Test coverage
- A zero or non-earlier `lastUpdated` preserves usage while setting the timestamp to the truncated current timestamp.

  ☐ Test coverage
- An earlier nonzero timestamp decays both directions over the same interval using their shared timestamp, with floored replenishment and the year-2106 assumption.

  ☐ Test coverage
- Decay preserves usage for zero inputs, clears it after a full window, and otherwise subtracts floored replenishment.

  ☐ Test coverage
- The function emits `RateLimitCheckpointed` before storing the replacement and emitting `RateLimitConfigSet`.

  ☐ Test coverage

Negative behavior

- The supplied concrete wrapper rejects non-owner callers before the library delegatecall.

  ☐ Negative test
- An enabled outbound direction with a zero limit or window reverts with `InvalidRateLimitConfig()` before checkpointing.

  ☐ Negative test
- An enabled inbound direction with a zero limit or window reverts with `InvalidRateLimitConfig()`, after outbound validation.

  ☐ Negative test
- Disabled directions accept zero fields, which preserve usage while advancing the timestamp; retained nonzero fields permit refill.

  ☐ Negative test
- Supplied nonpayable wrappers reject nonzero `msg.value` before authorization, although a payable delegatecaller could preserve value that this library ignores.

- ☐ Negative test

- Validation, delegatecall, or out-of-gas failure atomically rolls back all prior iterations, writes, and logs.
  - ☐ Negative test

Function call analysis

- `this._getRateLimiterStorage()`
  - This private Yul helper returns the fixed namespaced storage pointer and performs no external call.
- `this._checkpointRateLimit(_params[i].eid, this._effectiveRateLimitConfig(_params[i].eid)) -> this._getRateLimiterStorage()`
  - It writes and logs the state calculated under the old effective configuration without callbacks.
- `this._checkpointRateLimit(_params[i].eid, this._effectiveRateLimitConfig(_params[i].eid)) -> this._currentRateLimitState(_eid, _config) -> this._getRateLimiterStorage()`
  - It deterministically applies timestamp-based decay with floored replenishment and makes no external call.
- `this._effectiveRateLimitConfig(_params[i].eid) -> this._getRateLimiterStorage()`
  - It selects the stored override or default configuration before replacement. Zero inactive fields preserve usage; retained nonzero fields allow decay during the disabled interval and alter reenable history.
- **Manual call-path verification**: Supplied wrappers reject value and enforce `onlyOwner`. They invoke the fixed linked library by `DELEGATECALL`, preserving caller, value, address, and OFT/proxy storage, and the helpers make no further calls. ASI's `this.` traces omit validation, decay, wrappers, and delegatecall. Its unrelated builder failures occur at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48`.

Function: `setRateLimitGlobalConfig(RateLimiterModule.RateLimitGlobalConfig _globalConfig)`

This linked-library function overwrites caller-context configuration and emits `RateLimitGlobalConfigSet`; the declaration lacks authorization. Supplied nonpayable `onlyOwner` wrappers enforce restrictions that additional callers must preserve.

Inputs

- `_globalConfig`
  - **Control**: The caller selects `_globalConfig.isGloballyDisabled`, which supplied owner-only wrappers forward unchanged.
  - **Constraints**: ABI decoding requires a canonical boolean; supplied wrappers require owner authorization and zero value, while a payable linking caller could preserve nonzero value through `DELEGATECALL`.
  - **Impact**: The value overwrites caller-context global configuration, with `true` bypassing limits and `false` resuming preserved per-EID configuration and state.

Branches and code coverage

Intended branches

- With `_globalConfig.isGloballyDisabled == true`, the function writes `true` to the fixed slot and emits `RateLimitGlobalConfigSet(true)` from the effective delegating address.
  - ☐ Test coverage
- With `_globalConfig.isGloballyDisabled == false`, the function writes `false` to the fixed slot and emits `RateLimitGlobalConfigSet(false)` from the effective delegating address.
  - ☐ Test coverage
- The function overwrites either prior flag value without reading it.
  - ☐ Test coverage
- The `_getRateLimiterStorage()` helper selects slot `0xf02f50e1c246a23e457c592e128a403f9c1cc60baa3a1ad2dc69bb4a5a51be00` before assignment and event emission.
  - ☐ Test coverage
- Supplied wrappers follow the owner-gated wrapper → module setter → fixed linked-library `DELEGATECALL` path, including Tempo through inheritance.
  - ☐ Test coverage
- A payable linking caller can preserve outer `msg.value`, as established by 2/2 Solc 0.8.22 `consumeRateLimit` harness cases, though this is not setter branch coverage.
  - ☐ Test coverage

Negative behavior

- Malformed calldata, including a noncanonical ABI boolean, fails decoding before any write or event persists.

- ☐ Negative test

- A caller failing `onlyOwner` is rejected by supplied wrappers before delegation.
  - ☐ Negative test
- Nonzero value is rejected by supplied nonpayable wrappers, not by the library's delegatecall boundary.
  - ☐ Negative test
- A failed linked-library delegatecall propagates the revert and atomically rolls back the write and log.
  - ☐ Negative test
- A direct state-mutating `CALL` to deployed library bytecode is rejected by Solidity's library runtime guard.
  - ☐ Negative test

Function call analysis

- `this._getRateLimiterStorage()`
  - Source review shows this generated `this.` edge is incorrect; the function is a private pure internal Yul helper that only selects the fixed storage slot.
- **Manually identified execution boundary — entry into `RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)` via linked-library `DELEGATECALL`**: The fixed linked target preserves caller storage, address, sender, and value; the body performs no value movement, callback, creation, or reentrant external call.
- **Manual call-path verification**: Supplied nonpayable owner-only wrappers delegate through the nonvirtual module setter, while unrelated ASI builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol:174` and `FraxOFTUpgradeableTempo.sol:48` do not affect this setter path.

Function: `setRateLimitStates(RateLimiterModule.SetRateLimitStateParam[] _params)`

This linked-library function is invoked through a fixed `delegatecall`, so it executes in the caller's storage and address context. All provided ABI entry points are owner-gated, with no weaker overrides.

Inputs

- `_params`

- **Control**: The owner selects array length, order, EIDs, usages, and timestamps, while the fixed linked target cannot be changed at runtime.
- **Constraints**: ABI widths apply, future timestamps revert, zero timestamps require zero usages, and empty arrays, duplicate or zero EIDs, and usage above limits are accepted.
- **Impact**: Each element overwrites namespaced state and emits `RateLimitStateSet`, with duplicate EIDs emitting repeatedly and the last value determining storage.

Branches and code coverage

Intended branches

- An empty array completes without writes or logs.
  - ☐ Test coverage
- A nonzero timestamp not exceeding `block.timestamp` permits any usage values; the function stores the state and emits the corresponding event.
  - ☐ Test coverage
- A zero timestamp with both usages zero is valid and is stored and emitted.
  - ☐ Test coverage
- Duplicate EIDs are processed sequentially; each update emits an event, and the last state wins.
  - ☐ Test coverage
- Validation precedes each write and emission, with no intervening external interaction.
  - ☐ Test coverage

Negative behavior

- A non-owner is rejected by the supplied wrapper before library dispatch.
  - ☐ Negative test
- A timestamp greater than `block.timestamp` reverts with `InvalidRateLimitState()` before writing or emitting.
  - ☐ Negative test
- A zero timestamp with either usage nonzero reverts with `InvalidRateLimitState()` before writing or emitting.
  - ☐ Negative test
- Any later validation failure, linked-call failure, or out-of-gas condition atomically rolls back all writes and logs.
  - ☐ Negative test

- Nonzero value is rejected by supplied nonpayable wrappers, although `delegatecall` itself preserves any outer value.
    - ☐ Negative test
- Suite-level 6/6 results and the separate 2/2 payable harness do not establish per-branch coverage for this function.
    - ☐ Negative test

Function call analysis

- `this._getRateLimiterStorage()`
    - The FCA renders this private Yul helper as external, but the helper only assigns slot `0xf02f50e1c246a23e457c592e128a403f9c1cc60baa3a1ad2dc69bb4a5a51be00`. It cannot call or reenter.
- `RateLimiterLib.setRateLimitStates(_params)` — manually recovered concrete call omitted by the generated FCA
    - The module uses a fixed linked `delegatecall`, preserving sender and value without transferring value. It returns no application data and atomically propagates failure. A Solc 0.8.22 payable harness passed 2/2 for value preservation.
- **Manual call-path verification**: The path is owner-gated wrapper → `_setRateLimitStates` → fixed library `delegatecall` → storage-slot helper → timestamp validation → write → event. Validation makes no external call. Both directions share `lastUpdated`, so integer decay may discard subunit recovery. ASI builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` are outside this path.

5.27. Module: RemoteHopV201Tempo.sol

Function: `constructor(address _endpoint)`

This function deploys the implementation, queries the caller-selected endpoint for a nonzero native-token immutable, and disables implementation initializers. The endpoint and implementation are trusted; creator identity is unused, and proxy endpoint storage is initialized separately.

Inputs

- `_endpoint`
    - **Control**: The creator supplies the external target; `msg.sender` and `tx.origin` are unused, and nonzero `msg.value` is rejected.

- **Constraints**: The target must return an ABI-decodable nonzero address, but no endpoint identity, code-length, or returned-token code check is performed.
- **Impact**: The returned address becomes immutable `nativeToken`, while `_endpoint` is not written to the proxy's separate `HopV2Storage.endpoint` field.

Branches and code coverage

Intended branches

- A zero-value deployment against the valid Tempo endpoint returns runtime bytecode and supports proxy execution.
  - ☑ Test coverage (`RemoteHopV2O1TempoForkTest.testFork_RecoverERC20`, `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectToFraxt`
- Nonzero native-token response: a decoded nonzero token address is stored as immutable `nativeToken` and permits execution to continue.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct`

Negative behavior

- Native-value rejection: nonzero `msg.value` reverts because the constructor is nonpayable.
  - ☐ Negative test
- Constructor-calldata decoding failure: malformed `_endpoint` calldata prevents contract creation.
  - ☐ Negative test
- Endpoint-call failure: an endpoint revert or gas exhaustion aborts deployment atomically.
  - ☐ Negative test
- Endpoint return-decoding failure: missing, malformed, or otherwise invalid address return data — including from a zero or codeless target — reverts during ABI decoding.
  - ☐ Negative test
- Zero native-token response: a decoded zero address reverts with `NativeTokenUnavailable()`.
  - ☐ Negative test
- Initializer-lock failure: a revert from `_disableInitializers()` aborts creation and rolls back constructor-time effects.
  - ☐ Negative test

Function call analysis

- The call path `RemoteHopV2O1Tempo.constructor(_endpoint) -> TempoGasTokenBase.constructor(_endpoint) -> IEndpointV2Alt(_endpoint).nativeToken()` performs the sole external edge, a fixed-selector, zero-value staticcall whose decoded nonzero address initializes immutable `nativeToken`.
- On success, `RemoteHopV2O1Tempo.constructor -> _disableInitializers()` locks only the implementation's initializer state before runtime bytecode is returned; no token movement, approval, callback, or event occurs.
- **Manual call-path verification**: ASI omitted both constructor edges, but supplied `TempoGasTokenBase.sol` and the inherited OpenZeppelin routine establish the external `nativeToken()` query and subsequent `_disableInitializers()` call.

Function: `initialize(uint32 _localEid, address _endpoint, byte[32] _fraxtalHop, uint32 _numDVNs, address _EXECUTOR, address _DVN, address _TREASURY, address[] _approvedOfts)`

This function initializes proxy state once, grants the caller administration, stores configuration and approvals, and sets Fraxtal's remote hop. Only the `initializer` modifier restricts callers. The internal helper lacks `onlyInitializing`, downstream replay tracking excludes the GUID (reproduced 1/1), administrator transitions can disrupt queued settlement, and zero or unsupported configuration remains accepted (reproduced 2/2).

Inputs

- `_localEid`
  - **Control**: The first successful initializer caller supplies this value.
  - **Constraints**: It is `uint32` but otherwise unvalidated.
  - **Impact**: It controls routing and source-EID metadata.
- `_endpoint`
  - **Control**: The first successful initializer caller supplies this address.
  - **Constraints**: Code, nonzero, interface, and constructor-endpoint equality are unchecked.
  - **Impact**: It authorizes `lzCompose` and can block or compromise settlement.
- `_fraxtalHop`
  - **Control**: The caller supplies source `bytes32`, despite ASI displaying `byte[32]`.
  - **Constraints**: Nonzero, canonical-peer, and deployment validity are unchecked.
  - **Impact**: It configures `remoteHop[30_255]` for routing and trust classification.

- `_numDVNs`
  - **Control**: The first successful initializer caller supplies this value.
  - **Constraints**: It is `uint32` without minimum or maximum validation.
  - **Impact**: It scales DVN fees and may suppress, inflate, or overflow quotes.
- `_EXECUTOR`
  - **Control**: The first successful initializer caller supplies this address.
  - **Constraints**: Code, nonzero, interface, and allowlist checks are absent.
  - **Impact**: Misconfiguration can manipulate or revert later fee quotes.
- `_DVN`
  - **Control**: The first successful initializer caller supplies this address.
  - **Constraints**: Code, nonzero, interface, and allowlist checks are absent.
  - **Impact**: DVN quotes may fail or return arbitrary values.
- `_TREASURY`
  - **Control**: The first successful initializer caller supplies this address.
  - **Constraints**: Code, nonzero, interface, and allowlist checks are absent.
  - **Impact**: Treasury quotes may fail or be manipulated.
- `_approvedOfts`
  - **Control**: The caller controls array length, order, duplicates, and addresses.
  - **Constraints**: Empty, duplicate, zero, codeless, and incompatible entries are accepted.
  - **Impact**: Every element becomes trusted by send and compose paths.

The caller (`msg.sender`) receives admin, `msg.value` must be zero, and `tx.origin` is unused.

Branches and code coverage

Intended branches

- A first proxy initialization grants the deployer `DEFAULT_ADMIN_ROLE`, allowing subsequent administration.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- A new grant adds membership, enumeration, and `RoleGranted`; an existing grant does not.
  - ☐ Test coverage — no exact prepopulated-role initializer test is identified in the supplied verification.
- The loop skips empty arrays and idempotently processes duplicates.
  - ☐ Test coverage — no exact empty/duplicate/long-array initialization test is identified.

- Initialization stores sufficient scalar configuration, OFT approval, and Fraxtal peer state for direct and multihop sends.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectT` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_ToChainA_viaF`
- Exact assertions for every initialized scalar and approval remain absent.
  - ☐ Test coverage
- The nonpayable dispatcher admits only zero-`msg.value` calls.
  - ☐ Test coverage — no exact nonzero-value call to `initialize` is identified.
- Construction staticcalls `nativeToken()`, requires nonzero, and then disables implementation initialization.
  - ☐ Test coverage — no exact implementation-deployment/static-call or direct-initialization test is identified.

Negative behavior

- Repeated proxy and direct implementation initialization revert at the initializer guard.
  - ☐ Negative test — no exact repeated/direct initialization test is identified.
- An unintended eligible first caller receives admin and selects all configuration.
  - ☐ Negative test — no exact initialization-race test is identified.
- Invalid, inconsistent, duplicate, and unsupported configuration values are accepted.
  - ☐ Negative test — no exact invalid-configuration initializer test is identified.
- Any revert or out-of-gas condition atomically rolls back initialization.
  - ☐ Negative test — no exact initialization rollback test is identified.
- Nonzero native value is rejected before function execution.
  - ☐ Negative test — no exact nonzero-`msg.value` initializer test is identified.

Function call analysis

- `this.__init_HopV2O1(...) -> this._grantRole(...) -> this._getAccessControlEnumerableStorage()`
- `this.__init_HopV2O1(...) -> this._getHopV2Storage()`
- `this._setRemoteHop(HopV2O1Tempo.FRAXTAL_EID, _fraxtalHop) -> this._getHopV2Storage()`
- **Manual call-path verification**: These are internal storage paths without runtime external edges. Construction separately staticcalls the endpoint's `nativeToken()`, requires nonzero, stores it immutably, and disables initialization.

Function: `lzCompose(address _oft, byte[32], bytes _message, address, bytes)`

This endpoint-only payable settlement function validates the pause state, `_oft`, and replay state. Duplicate messages return early. Fresh messages are decoded, sanitized, transferred, optionally composed, and emitted as `Hop`. Successful calls retain unrecoverable `msg.value` because `recoverETH` always reverts.

Inputs

- `_oft`
  - **Control**: The endpoint supplies `_oft`; administrator-controlled approval determines acceptance.
  - **Constraints**: The `_oft` parameter must be approved and return a decodable token address when needed.
  - **Impact**: The `_oft` parameter namespaces replay, identifies events and callbacks, and selects the token.
- `<unnamed@1>`
  - **Control**: The endpoint supplies this `bytes32` GUID despite generated `byte[32]` rendering.
  - **Constraints**: The function accepts but never reads the GUID.
  - **Impact**: Excluding the GUID from replay tracking causes distinct GUIDs sharing the same replay tuple to collide.
- `_message`
  - **Control**: The endpoint supplies the envelope containing cross-chain payload data.
  - **Constraints**: Codec reads and fresh-message decoding must succeed; zero recipients remain allowed.
  - **Impact**: Its fields determine replay, trust, amount, recipient, callback, and metadata.
- `<unnamed@3>`
  - **Control**: The endpoint supplies this executor address.
  - **Constraints**: ABI decoding validates the address, but execution never reads it.
  - **Impact**: It has no execution effect.
- `<unnamed@4>`
  - **Control**: The endpoint supplies this executor data.
  - **Constraints**: ABI decoding locates the bytes, but execution never reads them.
  - **Impact**: It has no execution effect.

Branches and code coverage

Intended branches

- Validation requires the endpoint, unpaused state, and approved `_oft`.
  - ☐ Test coverage
- A duplicate returns before decoding or settlement while retaining `msg.value`.
  - ☐ Test coverage
- A fresh tuple is marked processed before emitting `MessageHash`.
  - ☐ Test coverage
- Fresh messages decode `HopMessage` and read the envelope's `amountLD`.
  - ☐ Test coverage
- Matching `remoteHop[srcEid]` and `composeFrom` preserves payload source metadata.
  - ☐ Test coverage
- Otherwise, envelope source EID and compose sender replace payload metadata.
  - ☐ Test coverage
- Recipients narrow to 160 bits; positive amounts trigger lookup, then transfer.
  - ☐ Test coverage
- Zero amounts skip token lookup and transfer but permit callbacks.
  - ☐ Test coverage
- Empty callback data skips `hopCompose`.
  - ☐ Test coverage
- Nonempty data calls `hopCompose` after any transfer, including zero amounts.
  - ☐ Test coverage
- Successful settlement emits `Hop` after all optional external interactions.
  - ☐ Test coverage

Negative behavior

- Non-endpoint callers revert with `NotEndpoint` before parsing or settlement.
  - ☐ Negative test
- Paused or unapproved execution reverts with `HopPaused` or `InvalidOFT`.
  - ☐ Negative test
- Malformed envelope or payload encoding reverts fresh replay marking.
  - ☐ Negative test
- Reverting or malformed positive-amount `token()` responses revert atomically.

- ☐ Negative test
- Failed or invalid SafeERC20 transfers revert before callback.
  - ☐ Negative test
- Reverting callbacks undo transfer, replay marking, events, and retained value.
  - ☐ Negative test
- Zero-equivalent recipients remain accepted, allowing token transfers or callbacks to the zero address.
  - ☐ Negative test
- Distinct GUIDs sharing the same replay tuple collide because GUID is excluded from replay tracking.
  - ☐ Negative test
- Queued credit can be affected by recovery, pre-marking, deapproval, pausing, and changes to the remote hop.
  - ☐ Negative test
- Successful fresh and duplicate calls retain `msg.value`; `recoverETH` always reverts.
  - ☐ Negative test

Function call analysis

- `this._validateComposeMessage(_oft, _message) -> this._getHopV2Storage()`
- `this._validateComposeMessage(_oft, _message) -> OFTComposeMsgCodec.srcEid(_message)`
- `this._validateComposeMessage(_oft, _message) -> OFTComposeMsgCodec.composeFrom(_message)`
- `this._validateComposeMessage(_oft, _message) -> OFTComposeMsgCodec.nonce(_message)`
- `OFTComposeMsgCodec.composeMsg(_message)`
- `OFTComposeMsgCodec.amountLD(_message)`
- `OFTComposeMsgCodec.srcEid(_message)`
- `OFTComposeMsgCodec.composeFrom(_message)`
- `this._sendLocal(...) -> IOFT(_oft).token() -> SafeERC20.safeTransfer(...)`
- `this._sendLocal(...) -> IHopComposer(recipient).hopCompose(...)`
  - Positive transfers follow `token()`; callbacks enable reentrancy, but same-tuple endpoint reentry returns as a duplicate.
- **Manual call-path verification**: Generated `this.*` calls are internal. The order is validation, replay, decoding, transfer, callback, and then `Hop`, without native forwarding.

Function: `sendOFT(address _oft, uint32 _dstEid, byte[32] _recipient, uint256 _amountLD, uint128 _dstGas, bytes _data)`

This permissionless payable entry point dust-floors principal then delivers locally or routes through Fraxtal using TIP-20 fees.

Inputs

- `_oft`
  - **Control**: The caller selects an administrator-approved OFT and its implementation.
  - **Constraints**: It must be approved, return a positive rate, and complete calls.
  - **Impact**: It controls flooring, tokens, allowances, routing, callbacks, and event identity.
- `_dstEid`
  - **Control**: The caller selects the destination EID.
  - **Constraints**: The local EID delivers synchronously; others route through Fraxtal.
  - **Impact**: It controls local delivery, remote routing, and second-hop quoting.
- `_recipient`
  - **Control**: The caller supplies the destination identifier.
  - **Constraints**: Zero and zero-decoded recipients remain accepted.
  - **Impact**: It selects the local callback, token recipient, or remote recipient.
- `_amountLD`
  - **Control**: The caller supplies the requested principal amount.
  - **Constraints**: Conversion-rate flooring applies, while zero cleaned principal remains permitted.
  - **Impact**: The cleaned amount controls collection, delivery, allowances, callbacks, and events.
- `_dstGas`
  - **Control**: The caller supplies destination gas.
  - **Constraints**: Composed routes enforce `400_000`; direct Fraxtal transfers ignore it.
  - **Impact**: It affects compose options and uncapped remote fees.
- `_data`
  - **Control**: The caller supplies unrestricted callback data.
  - **Constraints**: Empty Fraxtal data sends directly; other routes encode composition.
  - **Impact**: It controls callbacks, routing, and message size; quote sizing

overcounts multihop payload by 320 bytes and direct payload by one byte.

Branches and code coverage

Intended branches

- A zero-`msg.value` send continues successfully.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct`
- The nonzero-`msg.value` rejection remains untested.
  - ☐ Test coverage
- Unpaused execution continues; paused execution reverts with `HopPaused()`.
  - ☐ Test coverage
- An approved OFT proceeds through a complete direct send.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct`
- Direct `sendOFT` rejection of an unapproved OFT remains untested.
  - ☐ Test coverage
- A positive conversion rate cleans an exactly representable principal.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct`
- Flooring of dusty principal and zero-rate rejection remain untested.
  - ☐ Test coverage
- A nonzero cleaned principal is pulled from the sender.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct`
- The zero-cleaned-principal branch remains untested.
  - ☐ Test coverage
- The local EID selects `_sendLocal`; other EIDs select Tempo routing.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_SendOFTFeeInclusive_LocalSend_No` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectToFraxt`
- A local positive principal is transferred to the decoded recipient and empty data skips composition.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_SendOFTFeeInclusive_LocalSend_No`

- Local nonempty callbacks and zero-principal behavior remain untested.
  - ☐ Test coverage
- Empty-data Fraxtal sends directly; other remote routes encode compose data.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectT` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_ToChainA_viaF`
- Composed gas floors at `400_000`; Fraxtal compose gas is set to `1_000_000`.
  - ☐ Test coverage
- Local or final Fraxtal routing skips second-hop fees; others call `quoteHop`.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectT` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_ToChainA_viaF`
- Nonzero multipliers scale fees; zero means 1x with checked arithmetic.
  - ☑ Test coverage (`HopV2O1TempoIntegrationTest.test_Integration_SetFeeMultipliers`)
- Default options apply when absent; composition sizes messages as `360 + data.length`.
  - ☐ Test coverage
- Zero fees only resolve the token; nonzero fees require collection.
  - ☐ Test coverage
- Whitelisted tokens are pulled directly; other tokens require swap discovery.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectT` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectToFraxt` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_NonWhitelistedMultiHop_Quote`
- Discovery skips failed quotes and input tokens, choosing the cheapest candidate.
  - ☐ Test coverage
- Tested DEX fees fit `uint128`, and configurable bounded slippage padding lets a quote/fill-divergent send settle within its cap.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_QuoteStatic_NonWhiteli` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_SetFeeSwapSlippageBps_AdminO` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_SendOFT_NonWhitelisted_Survi`

- Saturation at `type(uint128).max` remains untested.
  - ☐ Test coverage
- DEX allowances reset after swaps; positive unused input is refunded.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_NonWhitelistedMultiHop` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_SendOFT_NonWhitelisted_Survi`
- The `setUserToken(paymentToken)` call executes on every remote route, including zero-fee routes.
  - ☐ Test coverage
- Distinct fee and principal tokens receive separate usable approvals, while matching tokens permit the OFT to debit principal plus its settled fee.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_Direct` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_frxUSD_Adapter_DirectToFraxt`
- The zero-fee approval-skip branch is not isolated.
  - ☐ Test coverage
- The `SendOFT` event is emitted only after all synchronous work succeeds.
  - ☐ Test coverage

Negative behavior

- Any nonzero native value is rejected because fees use TIP-20s.
  - ☐ Negative test
- Pausing or OFT deapproval rejects execution before flooring and transfer.
  - ☐ Negative test
- Invalid rates, call failures, arithmetic errors, unavailable swaps, or send failure revert.
  - ☐ Negative test
- Local transfer or callback failure reverses collection and suppresses emission.
  - ☐ Negative test
- Zero amounts, unsupported EIDs, zero recipients, oversized data, and affordable uncapped fees remain accepted.
  - ☐ Negative test
- Stateful external calls may reenter, relying on trusted callees and rollback.
  - ☐ Negative test

Function call analysis

- `this.removeDust(_oft, _amountLD) -> IOFT2(oft).decimalConversionRate()`
- `ITIP20(IOFT(_oft).token()).transferFrom(msg.sender, address(this), _amountLD)`
- `this._sendLocal(...) -> SafeERC20.safeTransfer(...) -> IHopComposer(recipient).hopCompose(...)`
- `this._generateSendParam(...) -> OptionsBuilder.newOptions() -> OptionsBuilder.addExecutorLzComposeOption(...)`
- `this.quoteHop(...) -> ILayerZeroDVN.getFee(...) -> IExecutor.getFee(...) -> ILayerZeroTreasury.getFee(...)`
- `this._collectNativeAltToken(...) -> TIP_FEE_MANAGER.userTokens(msg.sender) -> STABLECOIN_DEX.swapExactAmountOut(...)`
- `this._approveOftFee(...) -> ITIP20(paymentToken).approve(...) -> ITIP20(oftToken).approve(...)`
- `this._sendToDestination(...) -> TIP_FEE_MANAGER.setUserToken(paymentToken) -> IOFT(_oft).send{value: 0}(...)`
- External token, callback, quote, DEX, FeeManager, and OFT calls may reenter; callback failure reverses local delivery, while endpoint excess remains with the hop.
- **Manual call-path verification**: Generated `this.*` traces are internal artifacts. Synchronous failures roll back. Finalized asynchronous processing is unaffected.

Findings

- Direct fees are uncapped. Endpoint-dollar excess reaches the hop. Zero-decoded recipients remain accepted. Wrapper dust may fund fees despite frxUSD's current rate of one. Verified sizing overcounts by 320 bytes or one byte. Source-OFT replacement may collide with replay keys. Privileged postcredit actions may alter custody or settlement.
- Approved contracts, endpoint, FeeManager, DEX, quote providers, routing, administrators, and upgrades are trusted. Wrappers bound gross and minimum net amounts and become sender and funder.

Function: `setFeeSwapSlippageBps(uint16 _bps)`

This role-gated setter stores bounded fee-swap slippage in the fixed Tempo ERC-7201 slot and emits `FeeSwapSlippageBpsSet`. Stored zero is a sentinel whose getter returns 50 bps.

Inputs

- `_bps`

- **Control**: The caller supplies `_bps` as calldata `uint16`.
- **Constraints**: Values 0–200 are accepted, with zero reading as 50 and nonzero values unchanged.
- **Impact**: The value is stored and emitted and later pads nonwhitelisted-token quotes and maximum swap input.

- `msg.sender`
  - **Control**: The immediate caller controls `msg.sender`.
  - **Constraints**: The caller must hold `DEFAULT_ADMIN_ROLE`.
  - **Impact**: It authorizes the update but is neither stored nor emitted.
- `msg.value`
  - **Control**: The caller may attempt to attach native value.
  - **Constraints**: The nonpayable function accepts only zero value.
  - **Impact**: Nonzero value reverts before any persistent effect.
- `tx.origin`
  - **Control**: It identifies the transaction-originating account.
  - **Constraints**: No condition depends on it.
  - **Impact**: It is unused.

Branches and code coverage

Intended branches

- An authorized `_bps == 0` stores zero and makes the getter return the 50-bps default.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_QuoteStatic_NonWhiteli` `Remote-HopV2O1TempoRealOFTIntegration.test_Tempo_SetFeeSwapSlippageBps_AdminO`
- The zero-valued event is not asserted.
  - ☐ Test coverage
- Authorized `1 <= _bps < 200` stores, emits, and returns the exact value.
  - ☐ Test coverage
- Authorized `_bps == 200` accepts and stores the inclusive maximum.
  - ☑ Test coverage (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_SetFeeSwapSlippageBps_`
- Aggregate integration testing covered bounded configuration without exact-branch attribution.
  - ☐ Test coverage

Negative behavior

- A caller lacking `DEFAULT_ADMIN_ROLE` reverts with `AccessControlUnauthorizedAccount`.
  - ☑ Negative test (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_SetFeeSwapSlippageBps_`
- Unchanged storage and log state after the revert are not independently asserted.
  - ☐ Negative test
- A value above 200 reverts with `FeeSwapSlippageTooHigh`.
  - ☑ Negative test (`RemoteHopV2O1TempoRealOFTIntegration.test_Tempo_SetFeeSwapSlippageBps_`
- Absence of persistent effects is not independently asserted.
  - ☐ Negative test
- Malformed `uint16` calldata is rejected during ABI dispatch.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by nonpayable dispatch.
  - ☐ Negative test
- Any revert rolls back all state and logs from the transaction.
  - ☐ Negative test

Function call analysis

- `this._setFeeSwapSlippageBps(_bps) -> this._getTempoGasTokenStorage()`
  - Both calls are internal/private, and the Yul helper only binds the fixed slot `0xdde8387d01cc878517f663dabc81b0226de49746c19eb5f56ad6bbacf74af000`, so no external edge or reentrancy point exists.
- **Manual call-path verification**: The path is role check → internal cap check → fixed-slot write → event. Later quote, transfer, DEX-swap, and refund behavior uses the effective setting, while later direct-send execution lacks a caller-selected fee-token maximum.

5.28. Module: RemoteMintRedeemHop.sol

Function: `_mintRedeemViaFraxtal(address _oft, byte[32] _to, uint256 _amountLD)`

This internal function constructs the LayerZero `SendParam`, checks the aggregate fee, approves and sends the OFT, and then refunds excess ETH to `msg.sender`.

Inputs

- `_oft`
  - **Control**: Originates from the `mintRedeem` caller.
  - **Constraints**: Same as in `mintRedeem`.
  - **Impact**: Controls which OFT's `quoteSend` and `send` are called, and which token is approved.
- `_to`
  - **Control**: Derived from `msg.sender` by the only caller.
  - **Constraints**: Always `bytes32(uint256(uint160(msg.sender)))`.
  - **Impact**: Embedded in `composeMsg` and decoded on Fraxtal to identify the recipient.
- `_amountLD`
  - **Control**: Originates from the `mintRedeem` caller and is dust-stripped before reaching this function.
  - **Constraints**: Nonzero.
  - **Impact**: Both `amountLD` and `minAmountLD` in `SendParam` are set to this value, enforcing zero slippage tolerance.

Branches and code coverage

Intended branches

- The `quoteSend` and `quoteHop` fees are correctly summed and compared against `msg.value`.
  - ☐ Test coverage — the scoped tests do not assert the fee components
- Token approval is set exactly to `_amountLD`, and the OFT send consumes it fully.
  - ☐ Test coverage — the scoped tests do not assert the temporary allowance
- Excess ETH (`msg.value - finalFee`) is successfully returned to `msg.sender`.
  - ☑ Test coverage (`test_RemoteHop_mintRedeem`, `test_RemoteHop_mintRedeem2`)
- No excess ETH means no refund call is made.
  - ☐ Test coverage

Negative behavior

- The function reverts with `InsufficientFee` when `msg.value` is exactly one wei below `finalFee`.
  - ☐ Negative test
- The function reverts with `RefundFailed` when `msg.sender` is a contract that reverts on

ETH receipt.

- ☐ Negative test

- The function reverts when `IOFT.send` reverts (e.g., zero-slippage violation, endpoint unreachable).
  - ☐ Negative test

Function: `mintRedeem(address _oft, uint256 _amountLD)`

This is the primary user-facing entry point. A caller provides an OFT address and amount, pays an ETH fee, and bridges tokens to Fraxtal, where the mint/redeem logic executes via compose.

Inputs

- `_oft`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal `frxUsdOft` or `sfrxUsdOft`.
  - **Impact**: Determines which underlying token is pulled from the caller (`IOFT(_oft).token()`).
- `_amountLD`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be nonzero after dust removal via `decimalConversionRate()`.
  - **Impact**: Determines how many tokens are pulled from the caller and forwarded across the bridge.
- `msg.value` (implicit)
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be at least `IOFT(_oft).quoteSend(...) + quoteHop()` at transaction execution time.
  - **Impact**: Covers the LayerZero messaging fee.

Branches and code coverage

Intended branches

- Tokens are successfully transferred from the caller to this contract and then forwarded to Fraxtal via LayerZero.
  - ☐ Test coverage — the scoped tests do not assert principal transfer or packet forwarding
- Excess ETH beyond `finalFee` is refunded to the caller.

- ☑ Test coverage (`test_RemoteHop_mintRedeem`, `test_RemoteHop_mintRedeem2`)

- The `MintRedeem` event is emitted on success.
  - ☐ Test coverage
- Dust is correctly stripped, and the reduced amount is what gets bridged.
  - ☐ Test coverage

Negative behavior

- The function reverts with `HopPaused` when `paused` is `true`.
  - ☐ Negative test
- The function reverts with `InvalidOFT` when `_oft` is neither `frxUsdOft` nor `sfrxUsdOft`.
  - ☐ Negative test
- The function reverts with `ZeroAmountSend` when `_amountLD` is entirely dust (less than `decimalConversionRate`).
  - ☐ Negative test
- The function reverts with `InsufficientFee` when `msg.value` is less than the combined LayerZero and hop fee.
  - ☐ Negative test
- The function reverts with `RefundFailed` when the ETH refund to `msg.sender` fails.
  - ☐ Negative test
- The function reverts when the caller has not approved this contract to spend the underlying token.
  - ☐ Negative test

Function: `quoteHop()`

This function returns the total ETH fee for the "hop" leg by querying the DVN, EXECUTOR, and TREASURY.

Branches and code coverage

Intended branches

- The function returns `dvnFee * numDVNs + executorFee` plus the treasury surcharge, inflated by `hopFee` basis points.
  - ☐ Test coverage
- When `hopFee` is 0, the raw total is returned unchanged.

- ☐ Test coverage

- When `numDVNs` is 0, only executor and treasury fees are included.
  - ☐ Test coverage

Negative behavior

- The function reverts if any of `DVN`, `EXECUTOR`, or `TREASURY` revert (e.g., they are upgraded or decommissioned).
  - ☐ Negative test

Function: `recoverERC20(address tokenAddress, address recipient, uint256 tokenAmount)`

This is an owner-only function to rescue ERC-20 tokens held by this contract.

Inputs

- `tokenAddress`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Determines which token is transferred.
- `recipient`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Receives the recovered tokens.
- `tokenAmount`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Must not exceed the contract's balance of the token.
  - **Impact**: Determines how many tokens are transferred out.

Branches and code coverage

Intended branches

- Tokens are successfully transferred to `recipient`.
  - ☐ Test coverage

Function: `recoverETH(address recipient, uint256 tokenAmount)`

This is an owner-only function to rescue ETH held by this contract. The parameter name `tokenAmount` is misleading, as the value transferred is native ETH.

Inputs

- `recipient`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Receives the ETH.
- `tokenAmount`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Must not exceed the contract's ETH balance.
  - **Impact**: Determines how much ETH is sent.

Branches and code coverage

Intended branches

- ETH is successfully sent to `recipient`.
  - ☐ Test coverage

5.29. Module: RemoteMintRedeemHopTempo.sol

Function: `mintRedeem(address _oft, uint256 _amountLD, address _feeToken, uint256 _maxFeeTokenAmount)`

This is an explicit-token overload of `mintRedeem`. The caller names the TIP-20 to pay fees in and sets a maximum debit ceiling — the Tempo equivalent of the native-chain `msg.value` cap.

Inputs

- `_oft`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal `frxUsdOft` or `sfrxUsdOft`.
  - **Impact**: Determines the bridged token pulled from the caller.
- `_amountLD`
  - **Control**: Fully controlled by the caller.

    - **Constraints**: Dust-trimmed — reverts with `ZeroAmountSend` if trimmed to zero.
    - **Impact**: Amount of bridged token pulled from the caller and sent cross-chain.
- `_feeToken`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must not be `address(0)`.
    - **Impact**: The TIP-20 pulled from the caller to cover the LZ outbound fee and the retained return-leg estimate. Overrides the caller's FeeManager binding.
- `_maxFeeTokenAmount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: If the actual fee as reported by `quoteUserTokenFee` exceeds this, the call reverts with `FeeAboveCap` before any pull.
    - **Impact**: Upper bound on the fee debit in `_feeToken` units, analogous to the `msg.value` ceiling on native chains.

Branches and code coverage

Intended branches

- A cap below the live fee reverts with `FeeAboveCap` before any token is pulled.
    - ☑ Test coverage (`testFork_E2E_FeeCap_WhitelistedToken`)
- A cap exactly at the live fee succeeds.
    - ☑ Test coverage (`testFork_E2E_FeeCap_WhitelistedToken`)
- Explicitly naming a whitelisted token overrides a nonwhitelisted caller binding; only principal frxUSD and the 1:1 pathUSD fee are debited, and the hop binds and retains pathUSD.
    - ☑ Test coverage (`testFork_E2E_ExplicitWhitelistedTokenBypassesDex`)
- A DEX-routed explicit token overrides the caller's FeeManager binding, and the cap is checked against the padded DEX quote.
    - ☑ Test coverage (`testFork_E2E_FeeCap_DexRoutedToken`, `testFork_E2E_FrxUsd_FrxUsdFeeViaDex`)

Negative behavior

- The function reverts with `InvalidFeeToken` when `_feeToken` is `address(0)`.
    - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_MintRedeemRejectsZeroFeeToke`
- The function reverts with `HopPaused` when `paused` is `true`.
    - ☑ Negative test

(RemoteMintRedeemHopTempoForkTest.testFork_MintRedeemRespectsPause)

- The function reverts with InvalidOFT when _oft is neither frxUsdOft nor sfrxUsdOft.

  ☑ Negative test (RemoteMintRedeemHopTempoForkTest.testFork_MintRedeemRejectsUnapprovedO

- The function reverts with FeeAboveCap when _maxFeeTokenAmount is zero and the fee is nonzero.

  ☐ Negative test

- The function reverts with NoSwappableWhitelistedToken when _feeToken is not whitelisted and has no DEX route.

  ☐ Negative test

Function: mintRedeem(address _oft, uint256 _amountLD)

This function routes caller-specified OFT and amount through the Tempo fee path with no principal cap and no explicit fee token.

Inputs

- _oft
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal frxUsdOft or sfrxUsdOft.
  - **Impact**: Determines the bridged token pulled from the caller via IOFT(_oft).token().
- _amountLD
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Dust-trimmed via removeDust(_oft, _amountLD).
  - **Impact**: Amount of the bridged token pulled from the caller and sent cross-chain to Fraxtal.

Branches and code coverage

Intended branches

- A nonzero msg.value call is rejected.

  ☑ Test coverage (testFork_E2E_ValueCallProbe)

- The caller's pathUSD (default fee token) is pulled 1:1, and the hop retains quoteHop().

  ☑ Test coverage (testFork_E2E_FrxUsd_PathUsdFee)

- The caller's frxUSD is routed through the DEX; the net fee debit stays within the padded

quote, the DEX allowance is cleared, and no frxUSD remains on the hop.

- ☑ Test coverage (`testFork_E2E_FrxUsd_FrxUsdFeeViaDex`)

- sfrxUSD is bridged via a mintable OFT, with no approval stored and no residual allowance left on the hop.
  - ☑ Test coverage (`testFork_E2E_SfrxUsd_PathUsdFee`)
- The `FeeCollected` event is emitted with the correct send fee and retained amount.
  - ☑ Test coverage (`testFork_E2E_FrxUsd_PathUsdFee`)
- The `MintRedeem` event is emitted.
  - ☑ Test coverage (`testFork_E2E_FrxUsd_PathUsdFee`)
- The hop's FeeManager binding is updated to reflect the payment token used.
  - ☑ Test coverage (`testFork_E2E_FrxUsd_PathUsdFee`)
- Insufficient fee allowance reverts atomically without pulling principal.
  - ☑ Test coverage (`testFork_E2E_RevertsWithoutFeeAllowance`)
- An inherited stale binding cannot block a new caller (rebind-on-failure path).
  - ☑ Test coverage (`testFork_E2E_InheritedBindingCannotBlockNextCaller`)

Negative behavior

- The function reverts with `HopPaused` when paused.
  - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_MintRedeemRespectsPause`)
- The function reverts with `InvalidOFT` when `_oft` is neither `frxUsdOft` nor `sfrxUsdOft`.
  - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_MintRedeemRejectsUnapprovedO`
- The function reverts with `ZeroAmountSend` when `_amountLD` rounds to zero after `removeDust`.
  - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_MintRedeemRejectsDustOnlyAmo`

Function: `quoteUserTokenFee(address _oft, bytes32 _to, uint256 _amountLD, address _userToken)`

This off-chain and UI quote returns the amount of `_userToken` a caller must hold and approve before calling `mintRedeem`, given the OFT, recipient, amount, and fee token.

Inputs

- `_oft`

    - **Control**: Fully controlled by the caller.
    - **Constraints**: An address that does not implement `quoteSend` causes a revert.
    - **Impact**: Determines the LZ send fee (in LZD units) forwarded to `_quoteUserTokenFee` via `quote()`.
- `_to`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: Recipient encoded in the quoted cross-chain message.
- `_amountLD`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: Gross amount passed to `quote()`, which dust-trims it before the LZ quote — matching the trim performed at execution.
- `_userToken`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: A nonwhitelisted token with no DEX route reverts with `NoSwappableWhitelistedToken`.
    - **Impact**: Fee token whose required amount the function returns.

Branches and code coverage

Intended branches

- A whitelisted token returns the fee 1:1 with the LZ quote.
    - ☑ Test coverage (`testFork_E2E_FrxUsd_PathUsdFee`)
- A DEX-routed token returns the raw DEX quote padded by `feeSwapSlippageBps`.
    - ☑ Test coverage (`testFork_E2E_FrxUsd_FrxUsdFeeViaDex`)
- A `_userToken` of `address(0)` silently defaults to `PATH_USD`.
    - ☐ Test coverage

Negative behavior

- The function reverts with `NoSwappableWhitelistedToken` when `_userToken` is not whitelisted and has no DEX route.
    - ☐ Negative test

Function: `setFeeSwapSlippageBps(uint16 _bps)`

This is an owner-only setter for the slippage headroom applied to DEX-routed fee swaps. It controls how much extra input the hop pulls from the caller beyond the raw `quoteSwapExactAmountOut` figure to absorb per-order rounding on the StablecoinDEX.

Inputs

- `_bps`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Must not exceed `MAX_FEE_SWAP_SLIPPAGE_BPS` (200 bps).
  - **Impact**: Sets `_feeSwapSlippageBps`, which pads every subsequent DEX-routed fee quote and pull.

Branches and code coverage

Intended branches

- The owner stores the inclusive maximum value and emits `FeeSwapSlippageBpsSet(200)`.
  - ☑ Test coverage (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`
- Setting `_bps` to 0 stores 0, and `feeSwapSlippageBps()` returns `DEFAULT_FEE_SWAP_SLIPPAGE_BPS` (50).
  - ☑ Test coverage (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`

Negative behavior

- The function reverts with `OwnableUnauthorizedAccount` when called by a non-owner.
  - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`
- The function reverts with `FeeSwapSlippageTooHigh` when `_bps` exceeds `MAX_FEE_SWAP_SLIPPAGE_BPS` (200).
  - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`

5.30. Module: SFrxUSDOFTUpgradeable.sol

Function: `checkpointRateLimits(uint32[] _eids)`

This owner-only function checkpoints inbound and outbound usage for each EID under its effective configuration, writes the complete state and shared timestamp, and emits `RateLimitCheckpointed`. It returns nothing and moves no tokens or native value.

Inputs

- `_eids`
  - **Control**: The owner controls the ABI-valid `uint32[]`, including its length, order, duplicates, and members.
  - **Constraints**: Empty, duplicate, unordered, and unconfigured EIDs are accepted, with practical length bounded by calldata and gas.
  - **Impact**: Each occurrence selects its override or default configuration, checkpoints both usage directions in order, writes state, and emits one event.

Branches and code coverage

Intended branches

- Authorized callers reach the internal helper, which invokes the fixed linked library by delegatecall.
  - ☐ Test coverage
- An empty array skips all per-EID reads, writes, and events.
  - ☐ Test coverage
- A nonempty array processes every occurrence in order and emits one event per occurrence.
  - ☐ Test coverage
- An EID with `overrideDefaultConfig == true` uses its stored override configuration.
  - ☐ Test coverage
- An EID without an override, including an unconfigured EID, uses `defaultConfig`. Selected EIDs checkpoint old defaults, while omitted EIDs may later redecay under replacement defaults.
  - ☐ Test coverage
- A new, same-time, or nonincreasing narrowed timestamp preserves usage and stores the current `uint32` timestamp.
  - ☐ Test coverage
- A positive elapsed interval independently decays both directions and advances their shared timestamp.

- ☐ Test coverage
- Zero usage, limit, window, or elapsed time preserves that direction's usage.
  - ☐ Test coverage
- Elapsed time at least equal to the window reduces nonzero usage to zero.
  - ☐ Test coverage
- A partial interval applies floored proportional replenishment without increasing usage or overflowing the bounded operands.
  - ☐ Test coverage
- Global and directional enable flags do not gate decay, while zero limit or window preserves usage but still advances the timestamp.
  - ☐ Test coverage
- Both directions use one `lastUpdated`, materializing common-interval floor effects.
  - ☐ Test coverage
- The `block.timestamp` value is explicitly truncated to `uint32`, and operation after the wrap relies on the stated migration assumption.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts before the helper, delegatecall, writes, or events.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable dispatcher before modifier or body execution.
  - ☐ Negative test
- Empty, duplicate, unconfigured, and other ABI-valid `uint32` EIDs are not rejected.
  - ☐ Negative test
- Out-of-gas or delegatecall failure reverts atomically, removing all prior loop writes and logs.
  - ☐ Negative test

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The fixed linked-library edge executes by `DELEGATECALL` in the caller's storage and caller context, returns no value, and makes no callbacks or further external calls.
- **Manual call-path verification**: the generated `this._checkpointRateLimits` prefix is

incorrect because the source call is internal; the resolved nonvirtual helper uses the shared namespaced storage location and reaches no token, messaging, or virtual-hook dispatch.

Function: `constructor(address _lzEndpoint)`

Deployment forwards `_lzEndpoint` unchanged through the `OFTUpgradeable` constructor chain then disables initializers on implementation storage. Proxy storage remains unaffected. Endpoint correctness and unavailable imported-base behavior remain deployment trust assumptions.

Inputs

- `_lzEndpoint`
  - **Control**: The deployment transaction or factory selects `_lzEndpoint`.
  - **Constraints**: ABI decoding requires an `address`, but the leaf constructor performs no validation before forwarding it unchanged.
  - **Impact**: It selects the inherited OFT endpoint configuration, subject to any unknown imported-base behavior.
- `msg.sender`
  - **Control**: The account or factory creating the implementation determines `msg.sender`.
  - **Constraints**: The leaf constructor imposes no caller authorization or comparison.
  - **Impact**: The leaf body does not read or use `msg.sender`.
- `msg.value`
  - **Control**: The creator selects `msg.value` subject to available balance and creation mechanics.
  - **Constraints**: The nonpayable constructor requires `msg.value == 0`.
  - **Impact**: Successful construction neither forwards nor accounts for native value.
- `tx.origin`
  - **Control**: The originating transaction's EOA determines `tx.origin`.
  - **Constraints**: The leaf constructor imposes no condition on `tx.origin`.
  - **Impact**: The leaf body does not read or use `tx.origin`.

Branches and code coverage

Intended branches

- Base construction succeeds: the `_lzEndpoint` parameter is forwarded unchanged

through the statically selected `OFTUpgradeable` constructor chain.

- ☐ Test coverage

- Implementation lock succeeds: the `_disableInitializers()` function locks only the deployed implementation's initialization state, not proxy storage.
  - ☐ Test coverage
- Successful nonpayable entry: construction proceeds only with zero `msg.value`, while the leaf body ignores `msg.sender` and `tx.origin`.
  - ☐ Test coverage
- No leaf-level asset flow or dispatch: the leaf body performs no asset movement, endpoint call, callback, or child creation.
  - ☐ Test coverage

Negative behavior

- Nonzero native value: creation with `msg.value > 0` reverts because the constructor is nonpayable.
  - ☐ Negative test
- Malformed constructor arguments: creation data that cannot decode `_lzEndpoint` as an `address` cannot complete construction.
  - ☐ Negative test
- Inherited-constructor revert: any base-constructor revert aborts deployment and rolls back all construction-time effects.
  - ☐ Negative test
- Initializer-lock revert: any `_disableInitializers()` revert cancels creation and rolls back preceding base effects.
  - ☐ Negative test
- Invalid endpoint handling is not established by the leaf: the leaf adds no endpoint validation, while acceptance by the unavailable base constructor remains unknown.
  - ☐ Negative test

Function call analysis

- `constructor(address) -> OFTUpgradeable(_lzEndpoint)`
  - This is statically selected base construction, not a call to `_lzEndpoint`; the deployer-controlled address is forwarded unchanged, and any revert aborts creation.
- `constructor(address) -> _disableInitializers()`
  - This internal helper locks implementation storage without an

attacker-selected target, value movement, callback, or reentrancy edge, and failure rolls back deployment.

- **Manual call-path verification**: ASI emitted no generated FCA sink header; manual tracing confirms exactly the base-constructor step followed by `_disableInitializers()`.

Findings

- No verified constructor-specific vulnerability was identified. Endpoint correctness and any validation, state changes, emissions, or interactions inside the unavailable imported base constructor remain deployment trust assumptions.
- ASI parser/builder errors at `contracts/FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `contracts/FraxOFTUpgradeableTempo.sol@48:2180` concern sibling paths and do not omit either source-visible constructor step.

Function: `initializeV120()`

The `initializeV120()` function lets any caller migrate an eligible proxy's state to version 4, which writes fixed EIP-712 metadata and clears legacy hashes. Because callers can choose when to invoke it, signatures created for a different prior domain may become invalid.

Inputs

- Explicit parameters: None.
- Calldata: Only the selector is relevant; trailing calldata is unused.
- `msg.sender`: Any caller may trigger the migration while its life-cycle predicate remains open.
- `msg.value`: The nonpayable entry point requires zero native value.
- `tx.origin`: This value is unused.
- Initializer storage: The call is allowed only when initialization is inactive and the recorded version is below 4.
- Domain values: The `name()` and `version()` functions return the fixed values `"Staked Frax USD"` and `"1.2.0"`.
- Constants and storage sink: The version is 4, and EIP-712 storage uses slot `0xa16a46d94261c7517cc8ff89f61c0ce93598e3c849801011dee649a6a557d100`.
- Other environment values: The function does not read transaction, block, token, ownership, endpoint, or rate-limit state.

Branches and code coverage

Intended branches

- Eligible inactive proxy versions 0–3 write the fixed metadata, clear both hashes, record 4, exit initialization, and emit `Initialized(4)`.
  - ☐ Test coverage
- Target-local pure `name()` and `version()` calls dispatch internally without reading ERC-20 metadata storage.
  - ☐ Test coverage
- Constructor-disabled implementation storage does not disable an independently initialized proxy's migration path.
  - ☐ Test coverage
- The caller affects only timing, `tx.origin` is unused, and successful entry requires zero value.
  - ☐ Test coverage
- Later validation of permit and EIP-3009 signatures uses the rewritten domain, invalidating signatures created for a different prior domain.
  - ☐ Test coverage

Negative behavior

- Recorded version 4 or higher causes a pre-body revert without state changes.
  - ☐ Negative test
- Active initialization rejects the invocation without committing domain updates or logs.
  - ☐ Negative test
- Direct implementation calls revert because its constructor disabled initializers.
  - ☐ Negative test
- Nonzero native value is rejected before life-cycle or domain state can be committed.
  - ☐ Negative test
- Any execution revert rolls back initializer state, EIP-712 writes, and logs.
  - ☐ Negative test
- Inactive versions 0–3 all satisfy the below-4 predicate, which does not require version 3.
  - ☐ Negative test
- Non-admin callers are admitted when life-cycle conditions permit and can choose migration timing but not domain values.
  - ☐ Negative test

Function call analysis

- `this.__EIP712_init(this.name(), this.version()) -> this.__EIP712_init_unchained(name, version) -> this._getEIP712Storage()`
    - ASI's `this.*` labels misstate internal dispatch; the helpers write fixed namespaced storage, and the Yul body only assigns its storage pointer.
- **Manual call-path verification**: The target-local pure value producers and helper ordering were verified. There are no external calls, callbacks, value transfers, or reentrancy. ASI parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` concern unrelated sibling Tempo `_quote` expressions.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This owner-only function replaces the defaults for EIDs lacking active overrides, without checkpointing state or moving assets.

Inputs

- `_defaultConfig`
    - **Control**: The owner supplies the override flag, enabled flags, `uint32` windows, and `uint112` limits.
    - **Constraints**: Calldata must decode canonically, and each enabled direction requires a nonzero window and limit.
    - **Impact**: Success stores the full default configuration and emits its six non-override fields, affecting future capacity and recovery for default-using EIDs.

Branches and code coverage

Intended branches

- Owner-authorized valid replacement without an implicit checkpoint: the reproduction passed 1/1 and demonstrated that replacement without checkpointing exposed `1,000` units, versus `910` when checkpointed first.
    - ☐ Test coverage — audit-only reproduction, no scoped-repository test case was mapped
- Valid outbound configuration: disabled outbound accepts any typed values, while enabled outbound requires a nonzero limit and window.
    - ☐ Test coverage
- Valid inbound configuration: disabled inbound accepts any typed values, while enabled

inbound requires a nonzero limit and window.

- ☐ Test coverage

- Stored and emitted field mapping: success stores all seven fields and emits the six fields other than `overrideDefaultConfig`.
  - ☐ Test coverage
- Per-EID selection boundary: EIDs without overrides use the replacement default, while active per-EID overrides remain unaffected.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller other than `owner()` reverts before entering the function body.
  - ☐ Negative test
- ETH submission: any nonzero `msg.value` is rejected by the nonpayable entry point.
  - ☐ Negative test
- Malformed calldata: noncanonical or incomplete struct calldata reverts before mutation.
  - ☐ Negative test
- Invalid enabled outbound configuration: enabled outbound with a zero limit or window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Invalid enabled inbound configuration: enabled inbound with a zero limit or window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Linked-library execution failure: validation, delegatecall, or gas failure atomically preserves prior storage and suppresses emission.
  - ☐ Negative test

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`
  - The first hop is internal. The fixed linked library then executes by `DELEGATECALL`, validates, assigns namespaced storage, emits the event, and makes no further calls.
- **Manual call-path verification**: The verified path is external nonpayable setter → `onlyOwner` → internal nonvirtual `_setDefaultRateLimitConfig` → library `DELEGATECALL`; the generated `this.` prefix is erroneous, and proxy or direct context retains storage,

`address(this)`, sender, and zero value.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This owner-only batch setter checkpoints each EID then stores and emits validated future limits.

Inputs

- `_params`
  - **Control**: The owner supplies ordered EIDs, configurations, flags, windows, and limits.
  - **Constraints**: Enabled directions require nonzero limits and windows; duplicates and below-usage replacements remain allowed.
  - **Impact**: Entries execute sequentially, checkpoint usage, and store proposals; for duplicate EIDs, the last entry wins.
- `msg.sender`
  - **Control**: The immediate caller controls `msg.sender`.
  - **Constraints**: It must equal `owner()` before processing begins.
  - **Impact**: It authorizes the batch but is neither stored nor emitted.
- `owner()` and ownership storage
  - **Control**: Preexisting ownership storage determines the current owner.
  - **Constraints**: The caller must match that owner.
  - **Impact**: Ownership affects only authorization.
- `msg.value`
  - **Control**: The caller can attempt to attach native value.
  - **Constraints**: The nonpayable entry point rejects nonzero value.
  - **Impact**: No native value is forwarded or retained.
- `tx.origin`
  - **Control**: The originating transaction determines `tx.origin`.
  - **Constraints**: This path never reads it.
  - **Impact**: It has no effect.
- `block.timestamp`
  - **Control**: Consensus supplies the current timestamp.
  - **Constraints**: Checkpointing casts it to `uint32`, assuming migration occurs before timestamps begin truncating after 2106.
  - **Impact**: It advances `lastUpdated` and determines decay under prior effective configurations.
- Preexisting rate-limit storage

    - **Control**: Prior administration and consumption determine defaults, overrides, usage, and timestamps.
    - **Constraints**: Old state is checkpointed without revalidation; global disable does not suppress this setter.
    - **Impact**: Prior effective configurations determine checkpoints, including sequential duplicate-EID behavior.
- Current execution address and linked RateLimiterLib code
    - **Control**: Deployment fixes the execution context and linked library bytecode.
    - **Constraints**: Correct behavior assumes linkage to the supplied RateLimiterLib.
    - **Impact**: The `DELEGATECALL` mutates and emits from the executing contract or proxy.
- Available gas
    - **Control**: Transaction gas and batch size bound execution.
    - **Constraints**: No source-level batch-size cap exists.
    - **Impact**: Out-of-gas failure reverts the complete batch, including writes and logs.

Branches and code coverage

Intended branches

- When `msg.sender == owner()`, authorization succeeds and reaches the linked library.
    - ☐ Test coverage
- An empty array completes without storage writes or rate-limit events.
    - ☐ Test coverage
- Each direction validates when disabled or when its enabled limit and window are nonzero.
    - ☐ Test coverage
- Each entry checkpoints prior effective state then stores and emits its proposal.
    - ☐ Test coverage
- If `lastUpdated == 0` or is not earlier than the current block timestamp, usage persists and time advances.
    - ☐ Test coverage
- If `0 < lastUpdated < uint32(block.timestamp)`, inbound and outbound usage decay independently.
    - ☐ Test coverage
- Decay preserves zero usage, clears after full windows, and otherwise subtracts bounded replenishment.

- ☐ Test coverage

- True overrides activate per-EID values; false overrides store values while defaults remain effective.
  - ☐ Test coverage
- Distinct EIDs update distinct mapping keys atomically.
  - ☐ Test coverage
- Duplicate EIDs execute sequentially, emit both events, and leave the final proposal in storage.
  - ☐ Test coverage
- Limits below checkpointed usage are accepted, leaving zero availability until decay or reconfiguration.
  - ☐ Test coverage
- Zero placeholders prevent decay while checkpoints advance `lastUpdated`. Retained inactive values permit decay. Only `setDefaultRateLimitConfig` replaces defaults.
  - ☐ Test coverage

Negative behavior

- A non-owner caller reverts before helper or library execution.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by the nonpayable entry point.
  - ☐ Negative test
- Malformed calldata, noncanonical booleans, or oversized integers fail ABI decoding.
  - ☐ Negative test
- Enabled outbound limiting with a zero limit or window reverts before checkpointing.
  - ☐ Negative test
- Enabled inbound limiting with a zero limit or window reverts before checkpointing.
  - ☐ Negative test
- Oversized batches can exhaust gas during decoding, iteration, storage, or emission.
  - ☐ Negative test
- Validation, arithmetic, delegatecall, an explicit revert, or gas failure rolls back the entire batch.
  - ☐ Negative test
- Duplicates, inactive values, false overrides, below-usage limits, and unsafe owner values remain accepted.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
    - Generated `this.` implies a self-call that does not occur. The internal helper uses a fixed-library `DELEGATECALL`, preserves context, and propagates failures atomically.
- **Manual call-path verification**: The unique target resolves to the nonvirtual helper and supplied library. ASI builder failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` are unrelated.

Function: `setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)`

This owner-only setter replaces the executing token or proxy's global rate-limit configuration. Setting it to `true` bypasses later accounting and reports unlimited availability, while `false` restores configured enforcement without changing per-EID configuration or usage.

Inputs

- `_globalConfig`
    - **Control**: The authorized caller supplies `_globalConfig.isGloballyDisabled`, while `msg.sender` determines authorization. The linked-library target is fixed in the currently selected implementation bytecode, although a proxy upgrade may change that implementation outside this call's runtime inputs.
    - **Constraints**: Calldata must contain a canonical ABI boolean, `msg.sender` must be the current owner, and `msg.value` must be zero.
    - **Impact**: The value is stored in namespaced `RateLimiterStorage.globalConfig` and emitted without modifying per-EID configuration, usage, or token state.

Branches and code coverage

Intended branches

- An authorized call with `isGloballyDisabled == false` stores `false`, emits `RateLimitGlobalConfigSet(false)`, and enables configured enforcement.
    - ☐ Exact test coverage is not established by the supplied aggregate result
- An authorized call with `isGloballyDisabled == true` stores `true`, emits `RateLimitGlobalConfigSet(true)`, and bypasses subsequent inbound and outbound

accounting.

- ☐ Exact test coverage is not established by the supplied aggregate result

- The setter and library contain no input-dependent control flow; the boolean values produce distinct state outcomes only.
  - ☐ Exact test coverage is not established by the supplied aggregate result
- Authorization uses the current owner in the executing token or proxy's storage before invoking the helper.
  - ☐ Exact test coverage is not established by the supplied aggregate result

Negative behavior

- A non-owner call reverts before helper execution, preserving state and logs.
  - ☐ Negative-test coverage — it is not established by the supplied evidence
- Nonzero `msg.value` is rejected by nonpayable dispatch before the function body executes.
  - ☐ Negative-test coverage — it is not established by the supplied evidence
- Malformed calldata or a noncanonical ABI boolean reverts during decoding before execution.
  - ☐ Negative-test coverage — it is not established by the supplied evidence
- A failed linked-library delegatecall reverts atomically, rolling back the storage write and event.
  - ☐ Negative-test coverage — it is not established by the supplied evidence

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The source uses an internal helper rather than an external self-call, followed by a fixed linked-library `DELEGATECALL` that preserves caller context, writes configuration, emits the event, returns no value, and performs no callbacks or asset transfers.
- **Manual call-path verification**: The verified path is `setRateLimitGlobalConfig` → internal `RateLimiterModule._setRateLimitGlobalConfig` → `DELEGATECALL` to `RateLimiterLib.setRateLimitGlobalConfig` → storage write → event; the generated `this.` prefix is inaccurate, and unrelated ASI resolution errors do not alter this path.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

This owner-only setter replaces EID states in input order through the linked library and emits an event for each occurrence. For repeated EIDs, the last write prevails. Each timestamp becomes

both directions' decay reference.

Inputs

- `_params`
  - **Control**: The caller supplies the ordered EIDs, usages, and timestamps.
  - **Constraints**: Calldata must decode, future timestamps revert, and zero timestamps require zero usages.
  - **Impact**: Each element replaces its EID state and emits an event, while repeated EIDs leave the final value.
- `msg.sender`
  - **Control**: The immediate caller determines `msg.sender`, which delegatecall preserves.
  - **Constraints**: It must equal the current owner.
  - **Impact**: It controls authorization but is neither stored nor emitted.
- Current ownership state
  - **Control**: Prior initialization and ownership transitions determine the current owner.
  - **Constraints**: A valid owner must exist for authorization to succeed.
  - **Impact**: It gates entry without being modified.
- `msg.value`
  - **Control**: The caller selects the transaction value.
  - **Constraints**: The nonpayable entry point requires zero value.
  - **Impact**: No native value reaches the helper or library.
- `block.timestamp`
  - **Control**: The block environment supplies the current timestamp.
  - **Constraints**: Every supplied `lastUpdated` must be no greater than it.
  - **Impact**: It validates timestamps but is not automatically stored.
- Existing rate-limiter storage
  - **Control**: Earlier administration and consumption determine prior storage.
  - **Constraints**: Existing states and configurations are not consulted during validation.
  - **Impact**: Supplied EID states are replaced, while all other states and configurations remain unchanged.
- Execution context and linked RateLimiterLib code
  - **Control**: Deployment and upgrades select the active implementation and fixed linked library.
  - **Constraints**: The reviewed implementation and expected library bytecode

must be active.
- **Impact**: Delegatecall writes the calling contract or proxy's namespaced storage while preserving caller and emitter context.

- `tx.origin`
  - **Control**: The transaction originator determines `tx.origin`.
  - **Constraints**: This path never reads it.
  - **Impact**: It has no authorization, storage, event, or control-flow effect.

Branches and code coverage

Intended branches

- Authorized empty-array path: an authorized empty array succeeds without writes or events.
  - ☐ Test coverage
- Zero-timestamp zero-usage path: zero timestamp with both usages zero stores and emits the zero state.
  - ☐ Test coverage
- Nonzero past/current-timestamp path: any nonzero timestamp no later than the block time accepts representable usages.
  - ☐ Test coverage
- Usage above an effective limit: above-limit usage is accepted and may block later enabled consumption until capacity returns.
  - ☐ Test coverage
- Multiple distinct EIDs: distinct valid EIDs are written and emitted in input order.
  - ☐ Test coverage
- Repeated EIDs: every occurrence writes and emits, with the final occurrence prevailing.
  - ☐ Test coverage
- Target-specific call resolution: the target resolves through the inherited helper to the linked library without dispatching token or LayerZero paths.
  - ☐ Test coverage
- Coverage evidence assessment: reported module tests do not map assertions to this setter's individual branches.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a non-owner reverts before the helper executes.
  - ☐ Negative test

- Nonzero native value: any nonzero value fails the nonpayable dispatch check.
  - ☐ Negative test
- Invalid ABI encoding: malformed or out-of-range ABI data fails before the body executes.
  - ☐ Negative test
- Future timestamp: any future `lastUpdated` reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- Zero timestamp with nonzero outbound usage: this combination reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- Zero timestamp with nonzero inbound usage: this combination reverts with `InvalidRateLimitState`.
  - ☐ Negative test
- Invalid later element after a valid prefix: a later validation failure rolls back all earlier writes and logs.
  - ☐ Negative test
- No partial-success recovery path: authorization, decoding, validation, gas, or delegatecall failure reverts the entire transaction.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
  - The generated `this.` prefix does not match the source. The internal helper makes the sole external edge: a fixed linked-library delegatecall that returns no value. Under the expected bytecode, it only validates, writes, and emits; it has no callbacks or further calls.
- **Manual call-path verification**: The path is nonpayable decoding, `onlyOwner`, internal `RateLimiterModule._setRateLimitStates`, then `RateLimiterLib.setRateLimitStates`. ASI failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` occur only in sibling paths and leave the generated component graph incomplete.

5.31. Module: SfrxUSD2.sol

Function: `addMinter(address minter_address)`

After timelock authorization, the function activates a nonzero inactive address, appends it, emits `MinterAdded`, and grants later `minter_mint` and `minter_burn_from` authority. It returns no data.

Inputs

- `minter_address`
  - **Control**: The caller supplies the unchanged mapping key, array element, and event argument.
  - **Constraints**: It must be nonzero and have a false `minters` entry.
  - **Impact**: Success activates the address and grants downstream minter authority.
- `msg.sender` (implicit input)
  - **Control**: The immediate EVM caller sets `msg.sender`.
  - **Constraints**: The `_requireSenderIsTimelock()` function must accept it before any validation or effects.
  - **Impact**: It determines whether the addition may proceed.
- `msg.value` (implicit input)
  - **Control**: The caller supplies the transaction value.
  - **Constraints**: It must be zero because the function is nonpayable.
  - **Impact**: Nonzero value reverts before body execution.
- Current timelock authority (implicit inherited state input)
  - **Control**: Inherited `Timelock2Step` state determines the current authority.
  - **Constraints**: It must authorize `msg.sender` through `_requireSenderIsTimelock()`.
  - **Impact**: It gates every validation, write, and event.
- `minters[minter_address]` (implicit storage input)
  - **Control**: Prior additions and removals determine this mapping entry.
  - **Constraints**: It must be false after authorization and the nonzero check.
  - **Impact**: Success sets it true for later `onlyMinters` checks.
- `minters_array.length` (implicit storage input)
  - **Control**: Prior successful additions determine the current length.
  - **Constraints**: The append must complete within available gas.
  - **Impact**: Success increases array length, making later linear removal scans increasingly costly.
- `tx.origin` (implicit transaction context)
  - **Control**: The transaction originator determines `tx.origin`.
  - **Constraints**: None apply because the function does not read it.

- **Impact**: It has no effect on authorization, state, or events.

Branches and code coverage

Intended branches

- The `_requireSenderIsTimelock()` function accepts `msg.sender`, and execution continues, although the check does not prove that the configured authority enforces delayed governance.
  - ☐ Test coverage — the supplied suite totals do not map an individual test to this exact authorization-success branch
- A nonzero inactive address is activated, appended unchanged, emitted in `MinterAdded`, and returned with empty data.
  - ☐ Test coverage — no supplied test is mapped to this complete success path
- A removed address may be re-added as a new element without reusing its tombstone, causing permanent array growth and increasingly costly removal scans.
  - ☐ Test coverage — no supplied test is mapped to this re-addition path

Negative behavior

- Failed timelock authorization reverts before address validation, storage changes, or logging.
  - ☐ Negative test — no supplied test is mapped to this exact authorization-failure path
- A zero address reverts with "`Zero address detected`" before the mapping check or effects.
  - ☐ Negative test — no supplied test is mapped to this exact zero-address failure path
- An active address reverts with "`Address already exists`" before any write, append, or event.
  - ☐ Negative test — no supplied test is mapped to this exact duplicate-address failure path
- Nonzero `msg.value` triggers the nonpayable check before function-body execution.
  - ☐ Negative test — no supplied test is mapped to this exact nonpayable failure path
- Exceptional termination, including append out-of-gas, atomically rolls back all writes and the log.
  - ☐ Negative test — no supplied test is mapped to this exact exceptional-termination path

Function call analysis

- **Manual call-path verification**: The effective path is internal timelock authorization, local checks, mapping and array writes, then logging; descendants do not override the function, and no callback or reentrancy edge occurs.

Function: `burn(uint256 _amount)`

This public, nonpayable self-burn destroys only the sfrxUSD shares of `msg.sender`, emits `Transfer` and then `Burn`, pays no underlying, and returns nothing. Any caller may use it without owner, allowance, signature, timelock, minter, or `tx.origin` authorization.

Inputs

- `_amount`
  - **Control**: The caller supplies the unchanged `uint256` burn amount.
  - **Constraints**: Zero succeeds, positive amounts require sufficient caller balance, malformed calldata fails decoding, and nonzero `msg.value` is rejected.
  - **Impact**: Success reduces the caller's balance and total supply exactly, leaves underlying unchanged, and emits `Transfer` followed by `Burn`.

Branches and code coverage

Intended branches

- Positive amounts up to `balanceOf[msg.sender]` reduce the caller's balance and total supply then emit `Transfer` and `Burn`.
  - ☐ Test coverage
- A zero amount leaves balances and supply unchanged but still emits both zero-value events.
  - ☐ Test coverage
- Any address may burn only its own `msg.sender` balance, with no role or allowance branch.
  - ☐ Test coverage
- Successful execution destroys shares without transferring underlying tokens or native value.
  - ☐ Test coverage

Negative behavior

- Malformed or short calldata fails ABI decoding before any state change or event.

- ☐ Negative test

- Nonzero `msg.value` fails the nonpayable check before function execution.
  - ☐ Negative test
- An amount above the caller's balance reverts with arithmetic panic `0x11` and commits no writes or logs.
  - ☐ Negative test
- Using this function as a redemption destroys shares without paying underlying assets.
  - ☐ Negative test
- Changing `tx.origin` has no effect because only `msg.sender` selects the account and event subject.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: Source review confirms `SfrxUSD2.burn` → `super._burn` → `Solmate ERC20._burn`, with checked balance subtraction, unchecked supply subtraction, `Transfer` before `Burn`, empty return data, atomic rollback on failure, and no external call, callback, value movement, or reentrancy point.

Function: `constructor(IERC20 _underlying, string _name, string _symbol, address _timelockAddress)`

The constructor creates SfrxUSD2, initializing vault configuration, and rewards immutables, timelock state, and `_initialized`. Descendants fix metadata and use a zero timelock.

Inputs

- `_underlying`
  - **Control**: Direct creators select the address, while scoped descendants expose it as their only constructor argument.
  - **Constraints**: The first `decimals()` result must equal 18, and the second must permit checked computation of `10 ** decimals`.
  - **Impact**: It configures the inherited vault asset and immutable `UNDERLYING_PRECISION` without transferring assets or granting approval.
- `_name`
  - **Control**: Direct creators select the string, while scoped descendants fix it to `"Staked Frax USD"`.
  - **Constraints**: ABI decoding must succeed, and the supplied bodies impose no additional local validation.

    - **Impact**: It is forwarded as inherited vault/share-token name metadata.
- `_symbol`
    - **Control**: Direct creators select the string, while scoped descendants fix it to `"sfrxUSD"`.
    - **Constraints**: ABI decoding must succeed, and the supplied bodies impose no additional local validation.
    - **Impact**: It is forwarded as inherited vault/share-token symbol metadata.
- `_timelockAddress`
    - **Control**: Direct creators select the address, while scoped descendants fix it to `address(0)`.
    - **Constraints**: No local code, interface, nonzero-address, or delay-semantics check is visible.
    - **Impact**: It configures `Timelock2Step` on the created implementation, while an upgraded proxy preserves its existing storage.

Branches and code coverage

Intended branches

- Direct successful SfrxUSD2 creation: bases complete, decimal validation and precision computation succeed, immutables initialize, and `_initialized` becomes true.
    - ☐ Test coverage
- Scoped SfrxUSD3 creation: the selected underlying is forwarded with fixed metadata and a zero timelock.
    - ☐ Test coverage
- Scoped SfrxUSD creation: the underlying passes through SfrxUSD3 and reaches this constructor once.
    - ☐ Test coverage
- Successful creation return: completion installs runtime code and exposes the created address without a Solidity return value.
    - ☐ Test coverage
- Implicit-input behavior: the supplied constructor bodies do not read `msg.sender` or `tx.origin`, and zero `msg.value` is required. The imported base bodies are unavailable, and static decimal responses may depend on context.
    - ☐ Test coverage
- Upgrade implementation behavior: constructor storage affects only the implementation, while embedded immutables govern delegated proxy execution and must remain legacy-compatible.
    - ☐ Test coverage

Negative behavior

- Nonpayable boundary: nonzero `msg.value` reverts direct or descendant creation.
  - ☐ Negative test
- Argument-decoding boundary: malformed address or dynamic-string encoding reverts before construction completes.
  - ☐ Negative test
- Underlying-call failure: a reverting, gas-exhausting, or malformed `decimals()` response atomically aborts creation.
  - ☐ Negative test
- Unsupported decimals: a first result other than 18 or a second result causing exponentiation overflow reverts.
  - ☐ Negative test
- Base-constructor failure: any `ERC4626` or `Timelock2Step` constructor failure prevents `_initialized` from being set and runtime code from being installed.
  - ☐ Negative test
- No local metadata/timelock rejection: empty metadata and unchecked timelock addresses are not locally rejected; address equality proves neither code nor enforced delay.
  - ☐ Negative test

Function call analysis

- The `address(_underlying)` and `ERC20(address(_underlying))` expressions are nonexecuting casts. The `LinearRewardsErc4626_2(...)`, imported `ERC4626(...)`, and `Timelock2Step(...)` base constructors execute in the creation context. Two independent, valueless static `decimals()` calls validate 18 decimals and compute `UNDERLYING_PRECISION`. The `convert(ONE_YEAR)` call initializes `ONE_YEAR_UD60X18`, and `_initialized = true` is the final visible write. The SfrxUSD3 and SfrxUSD contracts add only constructor forwarding.
- **Manual call-path verification**: No generated FCA sink header was provided. Manual review confirmed the two static external decimal edges, immutable and timelock paths, descendant forwarding, atomic rollback, and absence of runtime callbacks during construction. The supplied constructors perform no token or ETH movement, approval, mint, burn, or event emission.

Function: `minter_burn_from(address b_address, uint256 b_amount)`

Authorized minters can burn any account's shares without allowance, holder consent, signature, timelock, or owner approval, provided the balance suffices. The nonpayable operation returns nothing, moves no underlying assets, and leaves pricing state unchanged.

Inputs

- `b_address`
  - **Control**: The authorized caller selects the share account without holder authentication.
  - **Constraints**: Its balance must cover `b_amount`, while `address(0)` is allowed when this condition holds.
  - **Impact**: Its share balance is reduced, and it is emitted as `from` in both events.
- `b_amount`
  - **Control**: The authorized caller selects the amount independently of allowances, pricing, `msg.value`, or `tx.origin`.
  - **Constraints**: It cannot exceed `balanceOf[b_address]`, but zero is permitted.
  - **Impact**: It exactly reduces `balanceOf[b_address]` and `totalSupply` and is emitted in both events.

Branches and code coverage

Intended branches

- Authorized positive burn: a positive amount within the selected account's balance reduces that balance and `totalSupply` then emits `Transfer` before `TokenMinterBurned`.
  - ☐ Test coverage
- Authorized zero-amount burn: a zero burn succeeds for any address, emits both zero-value events, and leaves balances and supply unchanged.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller absent from the active `minters` mapping reverts with `OnlyMinters()` before accounting or events.
  - ☐ Negative test
- Insufficient share balance: an amount exceeding the selected balance underflows in `_burn` and reverts all changes and logs.
  - ☐ Negative test
- Unexpected native value: Solidity rejects nonzero `msg.value` before the modifier or body executes.
  - ☐ Negative test

Minter revocation scalability: Tombstones and re-additions grow `minters_array` without bound. A late minter may become too costly to revoke without changing its current mapping authorization.

Function call analysis

- **Manual call-path verification**: The path SfrxUSD/SfrxUSD3 → `SfrxUSD2.minter_burn_from` → `Solmate ERC20._burn` has no callback or reentrancy edge, moves no underlying or native value, and only reduced supply can make supply-derived `totalAssets()` nonincreasing subject to rounding or zero price.

Function: `minter_mint(address m_address, uint256 m_amount)`

Only callers with `minters[msg.sender] == true` may mint shares without deposits, allowances, signatures, caps, or underlying-token movement. Minting changes supply-based `_previewTotalAssets` views despite unchanged underlying balances and does not modify authorization.

Inputs

- `m_address`
  - **Control**: The authorized minter selects the recipient without affecting authorization.
  - **Constraints**: It is passed unchanged, and the Solmate mint path accepts `address(0)`.
  - **Impact**: It selects the credited balance and appears as the recipient in both events.
- `m_amount`
  - **Control**: The authorized minter selects an uncapped amount without allowance or deposit requirements.
  - **Constraints**: Zero is accepted, supply addition is checked, and recipient-balance addition is unchecked.
  - **Impact**: It increases supply and the recipient balance while affecting supply-based asset views without moving underlying tokens.

Branches and code coverage

Intended branches

- When `minters[msg.sender] == true`, execution enters the body and internally invokes `super._mint(m_address, m_amount)`.
  - ☐ Test coverage
- If supply addition succeeds, supply and balance increase before `Transfer` and then `TokenMinterMinted` are emitted, with no return data or `Mint` event.
  - ☐ Test coverage

- An authorized mint to `address(0)` increases both total supply and `balanceOf[address(0)]` and emits both events.
  - ☐ Test coverage
- An authorized zero-amount mint leaves quantities unchanged but still emits both zero-value events.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts with `OnlyMinters()` before storage changes or event emission.
  - ☐ Negative test
- A `totalSupply + m_amount` overflow reverts before the unchecked balance update, leaving no state changes or events.
  - ☐ Negative test
- A call carrying nonzero `msg.value` is rejected before modifier or body execution because the function is nonpayable.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The path is `SfrxUSD2.minter_mint` → `onlyMinters` → internal Solmate `super._mint` → `Transfer` → `TokenMinterMinted`, with no callbacks or reentrancy point; permanent minter-history growth can make late-minter revocation exceed block gas, although `minters` remains authoritative.

Function: `removeMinter(address minter_address)`

After inherited timelock authorization, the function removes an active nonzero minter by deleting its mapping entry, tombstoning the first array match, and emitting `MinterRemoved`. It also succeeds when no array entry matches.

Inputs

- `minter_address`
  - **Control**: The caller selects the calldata value.
  - **Constraints**: Must be nonzero and active after authorization — no array match is required.
  - **Impact**: Selects the deleted mapping entry, optional first tombstone, emitted address, and scan cost.

- `msg.sender` (implicit input)
  - **Control**: Set by the immediate caller.
  - **Constraints**: Must pass inherited `_requireSenderIsTimelock()` before validation or mutation.
  - **Impact**: Determines whether execution reaches parameter validation and state changes.
- `msg.value` (implicit input)
  - **Control**: Selected by the caller at the call boundary.
  - **Constraints**: Must be zero because `removeMinter` is nonpayable.
  - **Impact**: Nonzero value reverts without changing state or logs.
- `tx.origin` (implicit input)
  - **Control**: Determined by the transaction originator.
  - **Constraints**: Not read by this function or a supplied descendant override.
  - **Impact**: Has no effect because authorization uses the immediate sender.
- `minters[minter_address]` (implicit storage input)
  - **Control**: Established by prior privileged minter administration.
  - **Constraints**: Must be true after sender and zero-address validation.
  - **Impact**: Authorizes removal, and deletion revokes later `onlyMinters` access.
- `minters_array` and `minters_array.length` (implicit storage inputs)
  - **Control**: Determined by historical additions and prior tombstoning removals.
  - **Constraints**: The loop scans from index zero, tombstones the first match, and permits no match.
  - **Impact**: They determine the optional tombstone location and linear gas cost of scanning the array.

Branches and code coverage

Intended branches

- Authorization succeeds when `_requireSenderIsTimelock()` accepts `msg.sender`.
  - ☐ Test coverage
- Validation succeeds when the address is nonzero and its mapping entry is true.
  - ☐ Test coverage
- Each nonmatching array entry is left unchanged before advancing the loop.
  - ☐ Test coverage
- The first matching array entry is tombstoned and `break` stops further scanning.
  - ☐ Test coverage

- If no array entry matches, deletion remains effective and the event is still emitted.
  - ☐ Test coverage
- Successful completion emits `MinterRemoved(minter_address)` and returns no data.
  - ☐ Test coverage

Negative behavior

- An unauthorized sender reverts before parameter validation or mutation.
  - ☐ Negative test
- A zero address reverts with `"Zero address detected"` before membership checking or effects.
  - ☐ Negative test
- An inactive address reverts with `"Address nonexistant"` before deletion, scanning, or emission.
  - ☐ Negative test
- Nonzero `msg.value` reverts because the function is nonpayable.
  - ☐ Negative test
- Scan exhaustion reverts atomically, which can prevent removal when the array is too large to scan within the available gas.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The path is SfrxUSD/SfrxUSD3 → `SfrxUSD2.removeMinter` → `_requireSenderIsTimelock()` → local checks/effects/event. The inherited check is internal, event emission invokes no code, later failure rolls back all effects, and the supplied source cannot establish that the configured timelock is a contract or enforces delay.

Function: `setAllPricingParams(uint256 _newPricePerShareStored, uint256 _newPricePerShareIncPerSecond, uint256 _newLastSync)`

This timelock-restricted setter atomically replaces three pricing fields, emits each supplied value, and performs no asset, share, or ETH movement. Authorization proves only equality with the configured timelock address, not that the address contains code or enforces a delay. Reviewed upgrades initialize it, while fresh empty proxies are unsupported.

Inputs

- `_newPricePerShareStored`
  - **Control**: Any `uint256` is accepted only from the active timelock authenticated through `msg.sender`.
  - **Constraints**: The value is stored directly without bounds, arithmetic, casting, or consistency checks.
  - **Impact**: It directly scales later pricing. Zero produces a zero computed price, and large values may cause PRBMath reverts.
- `_newPricePerShareIncPerSecond`
  - **Control**: Any `uint256` is accepted only from the active timelock authenticated through `msg.sender`.
  - **Constraints**: The value is stored directly without caps, nonzero requirements, arithmetic, or casting.
  - **Impact**: Zero disables time-based growth, while large values or elapsed intervals may later cause multiplication or PRBMath reverts.
- `_newLastSync`
  - **Control**: Any `uint256` may be proposed only by the active timelock authenticated through `msg.sender`.
  - **Constraints**: The value must be at most `block.timestamp`, while zero, past timestamps, and equality are accepted.
  - **Impact**: Older values enlarge the pricing exponent, while later previews using `_asOfTime < lastSync` may underflow.

Branches and code coverage

Intended branches

- When `msg.sender` is the active inherited timelock, authorization succeeds and timestamp validation proceeds.
  - ☐ Test coverage
- A past `_newLastSync` writes the stored price, rate, and timestamp, then emits their corresponding events in that order.
  - ☐ Test coverage
- A `_newLastSync` equal to `block.timestamp` succeeds because only strictly future timestamps are rejected.
  - ☐ Test coverage
- Authorized zero or arbitrary pricing values and zero or past timestamps are accepted without further validation.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts before timestamp validation, writes, or events, although the unreproduced `Timelock2Step` body prevents confirming the exact revert form.
  - ☐ Negative test
- An authorized future `_newLastSync` reverts with `MustNotBeInTheFuture()` before all writes and events.
  - ☐ Negative test
- Nonzero `msg.value` is rejected by Solidity's nonpayable dispatcher before body execution.
  - ☐ Negative test
- Any exceptional halt atomically rolls back all storage writes and logs.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: Concrete inheritance resolves to `SfrxUSD2.setAllPricingParams`, which performs nonpayable dispatch → internal `_requireSenderIsTimelock()` → timestamp check → three writes → three events → return with no external interaction or callback.

Function: `setPricePerShareIncPerSecond(uint256 _newPricePerShareIncPerSecond)`

After timelock authorization, this setter synchronizes price per share using the old pricing state, stores the new rate, and emits `SetPricePerShareIncPerSecond`. The synchronization return is ignored.

Inputs

- `_newPricePerShareIncPerSecond`
  - **Control**: The caller selects the full `uint256`.
  - **Constraints**: It is unrestricted but stored only after authorization and synchronization succeed.
  - **Impact**: It is stored unchanged, emitted, and controls future exponential growth.
- `msg.sender`
  - **Control**: The immediate caller supplies this value.
  - **Constraints**: It must match the configured timelock, although that address

check cannot prove the caller's code or delay semantics.
    - **Impact**: It controls authorization only.
- `msg.value`
    - **Control**: The caller may attempt to attach native value.
    - **Constraints**: The nonpayable dispatcher requires zero value.
    - **Impact**: Nonzero value reverts before any body execution.
- `tx.origin`
    - **Control**: The transaction originator determines this ambient value.
    - **Constraints**: The resolved path does not read it.
    - **Impact**: It has no effect.
- `block.timestamp`
    - **Control**: The block environment supplies the timestamp.
    - **Constraints**: Checked subtraction requires that it not precede `lastSync`.
    - **Impact**: It determines elapsed time and becomes the new `lastSync`.
- Existing pricing storage (`pricePerShareStored`, `pricePerShareIncPerSecond`, and `lastSync`)
    - **Control**: Prior synchronization and timelock setters establish these values.
    - **Constraints**: No scoped upper bound applies to the stored price or rate.
    - **Impact**: Old state exclusively determines synchronization, and failure rolls back all effects.

Branches and code coverage

Intended branches

- Authorized path: timelock authorization succeeds and execution reaches `sync()`.
    - ☐ Test coverage
- Zero-elapsed path: equal timestamps produce a zero exponent and retain the old price subject to fixed-point behavior.
    - ☐ Test coverage
- Positive-elapsed path: the old rate and elapsed time determine exponential growth.
    - ☐ Test coverage
- Synchronization-success path: the `sync()` function stores the computed price and current timestamp while its return is ignored.
    - ☐ Test coverage
- Completion path: the setter stores and emits the unchanged new rate.
    - ☐ Test coverage

Negative behavior

- Authorization failure reverts before synchronization, writes, or emission.
  - ☐ Negative test
- Nonzero `msg.value` is rejected before body execution.
  - ☐ Negative test
- A future `lastSync` causes checked subtraction to revert.
  - ☐ Negative test
- Overflow in `oldRate * elapsedTime` reverts before wrapping.
  - ☐ Negative test
- An unsupported PRBMath exponent causes `exp` to revert.
  - ☐ Negative test
- An invalid UD60x18 product causes `mul` to revert before synchronization writes.
  - ☐ Negative test

Function call analysis

- `this.sync() -> this._previewPricePerShare(block.timestamp) -> Casting.sol.wrap(this.pricePerShareIncPerSecond * _elapsedTime)`
  - This is internal computation using the old rate; checked multiplication may revert before pure `wrap`.
- `this.sync() -> this._previewPricePerShare(block.timestamp) -> Math.sol.exp(_exponentUD60_18)`
  - Internal pure `exp` derives growth from old state and time and may reject an excessive exponent.
- `this.sync() -> this._previewPricePerShare(block.timestamp) -> Casting.sol.unwrap(Math.sol.mul(Casting.sol.wrap(this.pricePerShareStored), _ePowUD60_18))`
  - Pure `unwrap` returns the nested multiplication result without narrowing.
- `this.sync() -> this._previewPricePerShare(block.timestamp) -> Math.sol.mul(Casting.sol.wrap(this.pricePerShareStored), _ePowUD60_18)`
  - Internal pure `mul` computes the synchronized price and may revert on invalid arithmetic.
- `this.sync() -> this._previewPricePerShare(block.timestamp) -> Casting.sol.wrap(this.pricePerShareStored)`
  - Pure `wrap` reinterprets the old stored price without scaling or external control flow.

- **Manual call-path verification**: The path has no external calls, callbacks, or reentrancy. Initialized proxy storage — not constructors — supplies active state. Generated `this.*` traces misrender internal calls and reads.

Function: `setPricePerShareStored(uint256 _newPricePerShareStored)`

This timelock-only setter records `lastSync = block.timestamp`, stores `_newPricePerShareStored` unchanged, and emits `SetPricePerShareStored`. The unvalidated value affects later pricing and vault accounting.

Inputs

- `_newPricePerShareStored`
  - **Control**: Calldata controls the full `uint256`, which reaches storage and the event after authorization.
  - **Constraints**: ABI decoding must succeed, but the function accepts every value from 0 through `type(uint256).max`.
  - **Impact**: The value is stored and emitted unchanged. It may cause later reverts or incorrect accounting.
- `msg.sender` (implicit)
  - **Control**: The immediate caller determines `msg.sender`, including a forwarding contract where applicable.
  - **Constraints**: The `_requireSenderIsTimelock()` function must recognize it as the active timelock.
  - **Impact**: It solely controls authorization and is neither stored nor emitted.
- `msg.value` (implicit)
  - **Control**: The immediate caller chooses the value sent with the call.
  - **Constraints**: Because the function is nonpayable, any nonzero value reverts before body execution.
  - **Impact**: No ETH is retained, transferred, or forwarded.
- `block.timestamp` (implicit)
  - **Control**: The block producer selects it within consensus constraints.
  - **Constraints**: The function applies no comparison, monotonicity check, arithmetic, or cast.
  - **Impact**: It is stored directly in `lastSync` as the baseline for later pricing calculations.
- `tx.origin` (implicit)
  - **Control**: The transaction's originating EOA determines it.
  - **Constraints**: The function does not read it.
  - **Impact**: It has no effect on authorization, storage, events, or interactions.

Branches and code coverage

Intended branches

- An authorized call writes the timestamp, stores the supplied price, emits the event, and returns no data.
  - ☐ Test coverage
- Authorized values of 0 and `type(uint256).max` follow the same path without validation, arithmetic, or casting.
  - ☐ Test coverage
- Every authorized success replaces `lastSync` with that block's timestamp before writing the price.
  - ☐ Test coverage
- The SfrxUSD3 and SfrxUSD contracts inherit this implementation without an override or added interaction.
  - ☐ Test coverage

Negative behavior

- An unauthorized caller reverts before either storage write or event emission.
  - ☐ Negative test
- A call with nonzero `msg.value` reverts before authorization or other effects.
  - ☐ Negative test
- Missing or undecodable parameter calldata reverts before body execution, while valid trailing calldata may be tolerated.
  - ☐ Negative test
- Zero, maximum, decreased, unbacked, or otherwise inconsistent prices are not rejected after successful authorization.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The `_requireSenderIsTimelock()` function is an inherited internal authorization helper. Resolution through SfrxUSD2 → SfrxUSD3 → SfrxUSD finds no override, callback, ETH/token movement, or reentrant edge. All reverts atomically roll back effects. Authentication does not prove that the configured timelock is a contract or enforces a delay, and the unavailable `Timelock2Step` source prevents verification of its exact error encoding and storage access.

5.32. Module: SfrxUSD3.sol

Function: `permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, byte[32] r, byte[32] s)`

This permissionless `permit` packs `r || s || v`, builds the EIP-712 digest while incrementing the nonce, validates authorization, and replaces the allowance. The preserved heading uses `byte[32]`, but the source uses the non–ABI-equivalent type `bytes32`.

Inputs

- `owner`
  - **Control**: Untrusted calldata selects the authorizing account independently of `msg.sender`.
  - **Constraints**: It must pass ECDSA recovery or ERC-1271 validation, and the zero address cannot satisfy the resolved path.
  - **Impact**: It selects the nonce, signer or external validation target, allowance owner, and first `Approval` topic.
- `spender`
  - **Control**: Untrusted calldata cryptographically bound into the authorization.
  - **Constraints**: It is forwarded unchanged and may be `address(0)`.
  - **Impact**: It selects the allowance recipient and second `Approval` topic without receiving assets.
- `value`
  - **Control**: Untrusted calldata cryptographically bound into the authorization.
  - **Constraints**: Any `uint256` value is accepted without local range checks.
  - **Impact**: It replaces `allowance[owner][spender]` and is emitted in `Approval`.
- `deadline`
  - **Control**: Untrusted calldata cryptographically bound into the authorization.
  - **Constraints**: Execution requires `block.timestamp <= deadline`, so equality is valid.
  - **Impact**: Expiry prevents nonce access, validation, approval, and event emission.
- `v`
  - **Control**: Untrusted `uint8` signature calldata.
  - **Constraints**: It is packed unchanged as the final byte of `r || s || v`.
  - **Impact**: It affects ECDSA or ERC-1271 acceptance without directly changing state.
- `r`

    - **Control**: Untrusted 32-byte signature calldata whose source type is `bytes32`.
    - **Constraints**: It is packed unchanged as the first 32 bytes of the 65-byte signature.
    - **Impact**: It affects signature acceptance and is exposed to a contract `owner`.
- `s`
    - **Control**: Untrusted 32-byte signature calldata whose source type is `bytes32`.
    - **Constraints**: It occupies signature bytes 32–63, and downstream signature validation enforces its validity.
    - **Impact**: It affects signature acceptance and is exposed to a contract `owner`.
- Implicit inputs are the timestamp, current nonce and allowance, chain/domain context, `address(this)`, and `owner` code; under proxy execution, the domain is bound to the proxy.
- The `msg.sender` and `tx.origin` values are unused, while nonzero `msg.value` is rejected because the function is nonpayable.

Branches and code coverage

Intended branches

- The seven-argument wrapper accepts the signed owner, spender, value, deadline, and `v`/`r`/`s` tuple and produces the expected allowance.
    - ☑ Test coverage (`TestSfrxUSDSignatures.test_Permit_succeeds`)
- Execution proceeds when `block.timestamp < deadline`.
    - ☑ Test coverage (`TestSfrxUSDSignatures.test_Permit_succeeds`)
- Execution also proceeds when `block.timestamp == deadline`.
    - ☐ Test coverage
- A successful first permit consumes its nonce, and a second permit signed with the next current nonce succeeds.
    - ☑ Test coverage (`TestSfrxUSDSignatures.test_Permit_succeeds`)
- EOA signature accepted: valid ECDSA authorization succeeds without an external call.
    - ☑ Test coverage (`TestSfrxUSDSignatures.test_Permit_succeeds`)
- ERC-1271 signature accepted: if direct ECDSA validation fails, validation may fall back to a staticcall to `owner` that returns the magic value.
    - ☐ Test coverage
- Successful validation replaces the allowance.
    - ☑ Test coverage (`TestSfrxUSDSignatures.test_Permit_succeeds`)
- Successful validation emits `Approval` and returns no ABI value.
    - ☐ Test coverage

Negative behavior

- Nonzero ETH: nonzero `msg.value` is rejected before the function body executes.
  - ☐ Negative test
- Expired deadline: a timestamp after `deadline` reverts before nonce access or external interaction.
  - ☐ Negative test
- Nonce overflow: a maximal nonce causes checked increment overflow and leaves state unchanged.
  - ☐ Negative test
- Invalid EOA signature: failure of both ECDSA and ERC-1271 validation reverts with `InvalidSignature` and rolls back the nonce.
  - ☐ Negative test
- ERC-1271 rejection or malformed response: ERC-1271 calls that fail or revert, or that return malformed, short, or nonmagic responses, are rejected.
  - ☐ Negative test
- ERC-1271 unusual control flow: the callback may consume gas or revert, but static context prevents persistent callback state changes.
  - ☐ Negative test
- Zero owner: the zero address cannot satisfy the resolved signature-validation path.
  - ☐ Negative test
- Zero spender: a valid authorization may set an allowance for `address(0)` because this is not a rejection branch.
  - ☐ Negative test

Function call analysis

- **Manual call-path verification**: The generated trace is incomplete because the internal permit path can conditionally `staticcall` the calldata-selected `owner` for ERC-1271 validation. ECDSA makes no external call, and any failure rolls back the nonce, allowance, and logs.

5.33. Module: TempoAltTokenBase.sol

Function: `constructor(address _lzEndpoint)`

This abstract constructor queries the endpoint and embeds the decoded native-token address in the implementation's immutable code, so proxy storage cannot modify it. Linked libraries, endpoints, precompiles, and permissioned tokens are trusted dependencies.

Inputs

- `_lzEndpoint`
  - **Control**: The deployer or factory supplies this address, and both concrete descendants forward it to this constructor and their LayerZero parent.
  - **Constraints**: No identity, code, interface, or nonzero check exists, but the query must return one canonically encoded address.
  - **Impact**: It selects the `nativeToken()` query target, whose result becomes the immutable used by later Tempo quote, approval, and wrapping paths.

Branches and code coverage

Intended branches

- With zero creation value and canonical endpoint return data, construction stores and later uses the returned native-token address.
  - ☑ Test coverage (`TempoFeeSwapSlippageTest.test_PayNativeAltToken_SettlesAndRefundsRema` `FraxOFTUpgradeableTem-poTest.test_AltEndpoint_PayNative_NoSwap_PathUsd`)
- A canonical `address(0)` return is also stored without constructor rejection.
  - ☐ Test coverage
- The FraxOFTUpgradeableTempo contract forwards `_lzEndpoint` to both FraxOFTUpgradeable and this constructor.
  - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_PayNative_NoSwap_PathUsd`)
- The FraxOFTMintableAdapterUpgradeableTIP20 contract forwards `_lzEndpoint` to both OFTAdapterUpgradeable and this constructor, while `_token` is not used in this query.
  - ☑ Test coverage (`FraxOFTMintableAdapterUpgradeableTIP20Test.test_LzSend_NoSwap_WhenUser`)
- The target may derive its return from `msg.sender`, which is the contract under construction, or from the original `tx.origin`, and no consistency check follows.
  - ☐ Test coverage

Negative behavior

- Each concrete descendant's nonpayable boundary rejects nonzero creation value before base construction.
  - ☐ Negative test
- A revert by the target, gas exhaustion, or another call failure atomically reverts creation.

- ☐ Negative test

- Empty, short, malformed, or noncanonical return data fails ABI decoding and atomically reverts creation.
  - ☐ Negative test
- A canonical zero address is accepted, but later nonzero-fee payment or quote paths may revert.
  - ☐ Negative test
- Any query or decoding failure rolls back all earlier effects and logs in the constructor sequence.
  - ☐ Negative test

Function call analysis

- Call path: `constructor(address _lzEndpoint)` → `IEndpointV2Alt(_lzEndpoint).nativeToken()`
  - Solidity performs one argumentless, valueless `STATICCALL`. The callee controls success, gas use, and return data. Successful decoding, casting, and immutable assignment do not write proxy storage or expose a runtime callback during construction.
- **Concrete descendant resolution and downstream taint**: Both concrete descendants invoke this base constructor once. They then use the immutable through TempoAltTokenLib for whitelist checks, swap discovery, fee quoting, approvals, and wrapping. The inherited `_quote` resolves before Tempo path validation.
- **Manual call-path verification**: ASI emitted no FCA sink header. Source review confirms the single `STATICCALL`. The parser failures at `FraxOFTMintableAdapterUpgradeableTIP20.sol@174` and `FraxOFTUpgradeableTempo.sol@48` concern descendant quote expressions, not another constructor sink.

5.34. Module: TempoAltTokenLib.sol

Function: `payNativeAltToken(ILZEndpointDollar _nativeToken, uint256 _nativeFee, address _endpointAddr, uint16 _slippageBps)`

This linked-library function pays fees directly or through exact-output swaps and returns early when the fee is zero.

Inputs

- `_nativeToken`

    - **Control**: Deployment fixes this immutable from the configured LayerZero endpoint.
    - **Constraints**: Nonzero fees require its whitelist and wrapping interfaces.
    - **Impact**: It controls accepted tokens, swap candidates, allowances, and fee crediting.
- `_nativeFee`
    - **Control**: Inherited `_lzSend` supplies the endpoint fee without recomputation.
    - **Constraints**: A zero fee returns early; swaps narrow output to `uint128`, while wrapping uses the full value.
    - **Impact**: Above-`uint128.max` values may revert or consume preexisting output-token custody.
- `_endpointAddr`
    - **Control**: Supplied paths use the deployment-fixed `address(endpoint)`.
    - **Constraints**: No independent validation occurs before forwarding it to `wrap`.
    - **Impact**: Misconfiguration may misroute or strand wrapped endpoint-native fees.
- `_slippageBps`
    - **Control**: The owner configures 1–200 bps; zero resolves to 50 bps.
    - **Constraints**: Padding floors percentages, adds needed headroom, and saturates at `uint128.max`.
    - **Impact**: Owner-configured slippage caps pulls and allowances, but callers cannot set an absolute maximum input.

Branches and code coverage

Intended branches

- Zero `_nativeFee` returns 0 before validation, calls, or movement.
    - ☐ Test coverage
- Fee resolution uses the payer's TIP-20 or `PATH_USD_ADDRESS`.
    - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_PayNative_NoSwap_PathUsd`, `TempoFeeSwapSlippageTest.test_PayNativeAltToken_SettlesAndRefundsRemainder`)
- Whitelisted input is pulled, approved, and wrapped without swapping.
    - ☑ Test coverage (`FraxOFTUpgradeableTempoTest.test_AltEndpoint_PayNative_NoSwap_PathUsd`)
- Route discovery selects the tested successful DEX route for the payer's nonwhitelisted token.

☑ Test coverage (TempoFeeSwapSlippageTest.test_PayNativeAltToken_SettlesAndRefundsRema

- Route discovery skips input and reverted quotes and chooses the cheapest of multiple successful candidates.

☐ Test coverage

- The actual library payment settles a quote/fill divergence within its default padded input cap.

☑ Test coverage (TempoFeeSwapSlippageTest.test_PayNativeAltToken_SettlesAndRefundsRema

- The actual library path exercises the one-unit headroom and `uint128.max` saturation branches.

☐ Test coverage

- DEX underspend clears residual allowance and refunds `msg.sender`.

☑ Test coverage (TempoFeeSwapSlippageTest.test_PayNativeAltToken_SettlesAndRefundsRema

- Output is wrapped for the exact native fee and delivered to the tested endpoint.

☑ Test coverage (TempoFeeSwapSlippageTest.test_PayNativeAltToken_SettlesAndRefundsRema FraxOFTUpgradeableTem-poTest.test_AltEndpoint_PayNative_NoSwap_PathUsd)

- The library return value is asserted to be zero.

☐ Test coverage

- The `TempoFeeSwapSlippageTest` suite passed 9/9 for slippage, rounding, refunds, and conservation.

☐ Test coverage

Negative behavior

- Nonzero fees with zero `_nativeToken` revert with `NativeTokenUnavailable`; zero bypasses validation.

☐ Negative test

- If no candidate route succeeds, the call reverts with `NoSwappableWhitelistedToken(userToken)`.

☐ Negative test

- Fee-manager and whitelist failures propagate; individual DEX quote failures are skipped.

☐ Negative test

- Uncaught token, DEX, wrapping, or endpoint failures revert earlier effects.

☐ Negative test

- Above-`uint128.max` fees truncate swap output but retain full wrapping amounts.
  - ☐ Negative test
- Mutable token, DEX, and endpoint-dollar calls may call back without guards.
  - ☐ Negative test
- Both supplied `send` entry points reject nonzero `msg.value`.
  - ☑ Negative test (`FraxOFTUpgradeableTempoTest.test_Send_RevertsWhenMsgValueNonZero`, `FraxOFTMintableAdapterUpgradeableTIP20Test.test_Send_RevertsWhenMsgValueNonZero`)
- Rejection before invoking this library is not independently instrumented.
  - ☐ Negative test

Function call analysis

- `_resolveUserToken() -> StdPrecompiles.TIP_FEE_MANAGER.userTokens(msg.sender)`
- `_nativeToken.isWhitelistedToken(userToken)`
- `ITIP20(userToken).transferFrom(msg.sender, address(this), _nativeFee)`
- `ITIP20(userToken).approve(address(_nativeToken), _nativeFee)`
- `_nativeToken.wrap(userToken, _endpointAddr, _nativeFee)`
- `findSwapTarget(...) -> _nativeToken.getWhitelistedTokens()`
- `findSwapTarget(...) -> StdPrecompiles.STABLECOIN_DEX.quoteSwapExactAmountOut(...)`
- `ITIP20(userToken).transferFrom(msg.sender, address(this), maxAmountIn)`
- `ITIP20(userToken).approve(address(StdPrecompiles.STABLECOIN_DEX), maxAmountIn)`
- `StdPrecompiles.STABLECOIN_DEX.swapExactAmountOut(...)`
- `ITIP20(userToken).approve(address(StdPrecompiles.STABLECOIN_DEX), 0)`
- `ITIP20(userToken).transfer(msg.sender, maxAmountIn - spentAmountIn)`
- `ITIP20(targetToken).approve(address(_nativeToken), _nativeFee)`
- `_nativeToken.wrap(targetToken, _endpointAddr, _nativeFee)`
- External edges trust the fee manager, TIP-20s, DEX, endpoint-dollar, and endpoint; downstream excess refunds remain denominated in `LayerZeroEndpointDollar`.
- **Manual call-path verification**: Compiler-generated `DELEGATECALL` preserves caller, OFT context, storage, and value. The body writes no storage or events. A Solc 0.8.22 payable harness passed 2/2 without a delegatecall value guard. ASI failures `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` are nested virtual-`super` parser failures. Generated `this.*` traces imply external self-calls and omit delegation, construction, quote, debit, endpoint, and event paths.

5.35. Module: TempoFeeInclusiveWrapper.sol

Function: `constructor(address _hop)`

This function deploys an ownerless wrapper, initializes the inherited reentrancy guard, and immutably stores a nonzero coded `_hop` without calling it. Code presence proves neither interface compatibility nor identity, and tokens transferred outside the intended flow are unrecoverable.

Inputs

- `_hop`
  - **Control**: The creating account or factory selects this untrusted deployment input.
  - **Constraints**: Zero or codeless addresses revert with `InvalidHop(_hop)`, while any nonzero address with code passes.
  - **Impact**: The unchanged address becomes immutable `HOP` and later controls validation, quotes, allowance use, reverts, and bridging behavior.
- `msg.sender`
  - **Control**: The immediate creator may be an EOA or factory contract.
  - **Constraints**: No authorization or identity check is applied.
  - **Impact**: It establishes no ownership, role, recovery authority, configuration, or fund movement.
- `msg.value`
  - **Control**: The immediate creator selects the value attached to contract creation.
  - **Constraints**: Solidity's nonpayable constructor logic requires `msg.value == 0`.
  - **Impact**: Any nonzero value reverts creation atomically, while zero value has no further effect.
- `tx.origin`
  - **Control**: The transaction-originating EOA controls this implicit value.
  - **Constraints**: It is never read or checked.
  - **Impact**: It affects no authorization, guard initialization, immutable configuration, or interactions.

Branches and code coverage

Intended branches

- With zero creation value and a coded nonzero hop, deployment succeeds and `HOP` stores the exact address.

- ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_Constructor_RejectsZeroOrCodelessHop`)
- Absence of events and fund movement is not asserted.
  - ☐ Test coverage
- For nonzero `_hop`, code length is inspected and only a nonzero result reaches immutable assignment.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_Constructor_RejectsZeroOrCodelessHop`)
- The `msg.sender` and `tx.origin` values are unused, so no creator authorization or identity persistence occurs.
  - ☐ Test coverage

Negative behavior

- A zero `_hop` reverts with `InvalidHop(address(0))`.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_Constructor_RejectsZeroOrCodelessHop`)
- Short-circuit evaluation itself is not instrumented.
  - ☐ Negative test
- A nonzero but codeless `_hop`, including an EOA or undeployed address, reverts with `InvalidHop(_hop)`.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_Constructor_RejectsZeroOrCodelessHop`)
- Nonzero `msg.value` reverts construction and rolls back all constructor-chain effects.
  - ☐ Negative test
- Any nonzero address with code is accepted despite possible interface incompatibility, proxy upgrades, or malicious behavior.
  - ☐ Negative test

Function call analysis

- The ASI evidence contains no generated FCA sink header for this constructor; inherited `ReentrancyGuard()` initialization runs internally, while `_hop.code.length` and `IRemoteHopTempo(_hop)` execute no target code and create no callback or reentrancy point.
- **Manual call-path verification**: Source review confirms no nested creation or external, static, delegate, low-level, token, native-value, or precompile call occurs, and all failures revert guard initialization and immutable assignment atomically.

Findings

- Deployment must trust the selected hop and its downstream OFTs, endpoint, fee, DEX, bridging, upgrade, and administrative configuration; the ownerless wrapper cannot recover off-flow token transfers.
- Generated first-pass “sole effect” wording omitted inherited `ReentrancyGuard` initialization; manual source review corrected the constructor-chain trace, and no constructor-specific test evidence marks the coverage items above as executed.

Function: `sendOFTFeeInclusive(address _oft, uint32 _dstEid, byte[32] _recipient, uint256 _maxAmountInLD, uint256 _minNetAmountLD, uint128 _dstGas, bytes _data)`

This permissionless, ownerless, `nonReentrant` bridge entry point enforces a dust-cleaned net amount, a gross input cap, a minimum net floor, and delta-only refunds.

Inputs

- `_oft`
  - **Control**: The caller selects a hop-approved OFT.
  - **Constraints**: Its token and fee route must support invoked operations.
  - **Impact**: It determines tokens, dust conversion, quoting, and delivery.
- `_dstEid`
  - **Control**: The caller selects the destination endpoint.
  - **Constraints**: The wrapper validates no destination, peer, or route.
  - **Impact**: It selects local, direct Fraxtal, or multihop delivery.
- `_recipient`
  - **Control**: The caller supplies the destination recipient.
  - **Constraints**: Zero and destination-specific encodings remain unchecked.
  - **Impact**: Local delivery truncates; remote delivery preserves all bytes.
- `_maxAmountInLD`
  - **Control**: The caller sets the gross token budget.
  - **Constraints**: It must exceed fees and yield nonzero cleaned net.
  - **Impact**: It caps pulls, allowance, principal, and fees.
- `_minNetAmountLD`
  - **Control**: The caller sets the minimum acceptable net.
  - **Constraints**: It must be nonzero and not exceed previewed net.
  - **Impact**: It rejects an unacceptable live quote before funding.
- `_dstGas`

    - **Control**: The caller supplies destination gas.
    - **Constraints**: The wrapper imposes no explicit bound.
    - **Impact**: Preview and execution forward or normalize it.
- `_data`
    - **Control**: The caller supplies dynamic compose data.
    - **Constraints**: It must be empty.
    - **Impact**: Emptiness prevents outbound application composition.
- `msg.sender`
    - **Control**: Any EOA or contract may call.
    - **Constraints**: The caller must fund, approve, and accept refunds.
    - **Impact**: It is debited, refunded, and emitted.
- `msg.value`
    - **Control**: The caller controls attached native value.
    - **Constraints**: It must equal zero.
    - **Impact**: It funds no fees, calls, or refunds.
- `tx.origin`
    - **Control**: The transaction origin is implicit and unused.
    - **Constraints**: No call-path condition depends upon it.
    - **Impact**: It affects no authorization, routing, accounting, or event.

Branches and code coverage

Intended branches

- Standard success previews, pulls, sends, clears allowance, and refunds deltas.
    - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_HappyPath`, `TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_LeavesNoAllowanceToTheHop`, `TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RefundsSubDustRemainder`)
- Local delivery transfers twice-cleaned principal to the low-160-bit recipient.
    - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_SendOFTFeeInclusive_LocalSend_No`
- Fraxtal delivery sends directly without second-leg composition.
    - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_SendOFTFeeInclusive_EndToEnd`)
- Multihop delivery embeds `HopMessage`; quote sizing prices 360 bytes for multihop delivery versus 40 bytes for direct delivery.

- ☐ Test coverage
- Zero fees skip both wrapper FeeManager calls.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_SkipsUserTokenWr`, `TempoFeeInclusiveWrapperFork-Test.test_SendOFTFeeInclusive_LocalSend_NoFeeNoFeeManagerBinding`)
- Matching fee-token bindings skip the FeeManager write.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_SetsUserTokenOnl`
- A different fee-token binding is updated for successful sends, and a rejected binding leaves caller funding untouched.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_SetsUserTokenOnl`, `TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_SetsUserTokenWhenBoundToAnotherToken`, `TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_FeeManagerRejectionPrecedesThePull`)
- Whitelisted fee tokens avoid DEX swaps.
  - ☐ Test coverage
- The real nonwhitelisted bridged fee token uses a viable bounded DEX route, and the padded implementation refunds unspent headroom.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperForkTest.test_SendOFTFeeInclusive_EndToEnd`, `TempoFeeInclusiveWrapperPaddedHopFork-Test.test_SendOFTFeeInclusive_EndToEnd`)
- Cheapest selection among multiple viable routes is not isolated.
  - ☐ Test coverage
- An unchanged ending balance produces no refund.
  - ☐ Test coverage
- A balance increase refunds only the current call's delta; preexisting wrapper balance is excluded and left untouched.
  - ☑ Test coverage (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RefundsSubDustRe`, `TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_EmitsTheFeeActuallyTaken`, `TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_RefundExcludesPreExistingBalance`, `TempoFeeInclusiveWrapperFork-Test.test_SendOFTFeeInclusive_PreExistingBalanceIsUntouched`, `TempoFeeInclusiveWrapperPaddedHopFork-`

Test.test_SendOFTFeeInclusive_EndToEnd)

- Gross above fees is dust-cleaned and floor-checked.
  - ☑ Test coverage (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RefundsSubDustRe TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_SucceedsWhenNetEqualsMinNet, TempoFeeInclusiveWrap-perTest.testFuzz_SendOFTFeeInclusive_ConservesCallerFunds)
- Gross not above fees yields zero previewed net and reverts.
  - ☑ Test coverage (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsWhenFeeEx TempoFeeInclusiveWrap-perTest.test_QuoteFeeInclusive_ReturnsZeroNetWhenFeeExceedsInput)

Negative behavior

- The nonReentrant modifier blocks recursion while quotes remain callable.
  - ☐ Negative test
- Nonzero msg.value triggers MsgValueNotZero.
  - ☑ Negative test (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsWhenMsgVa
- Zero gross triggers ZeroAmount.
  - ☑ Negative test (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsOnZeroAmo
- Zero minimum net triggers ZeroMinNetAmount.
  - ☑ Negative test (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsOnZeroMin TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_EchoedZeroQuoteCannotDisableTheFloor)
- Nonempty _data triggers ComposeNotSupported.
  - ☑ Negative test (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsOnCompose
- A paused hop aborts before funding, and a later hop send revert rolls back all funding.
  - ☑ Negative test (TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsWhenHopPa TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_HopRevertRollsBackEverything, TempoFeeInclusiveWrapperFork-Test.test_HopPaused_QuoteAndSendFailClosed)
- Malformed hop return data remains untested.

- ☐ Negative test

- Unapproved OFTs abort before funding.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsOnUnappro` `TempoFeeInclusiveWrapperFork-Test.test_UnapprovedOft_QuoteAndSendFailClosed`)
- Reverting or malformed OFT approval checks also abort.
  - ☐ Negative test
- Failed OFT token lookup aborts.
  - ☐ Negative test
- Quote failures abort; unusable DEX candidates are skipped.
  - ☐ Negative test
- Invalid conversion or dust responses abort preview.
  - ☐ Negative test
- Fees at or above the gross input trigger `FeeExceedsInput`.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsWhenFeeEx`
- Zero or below-floor cleaned net reverts before funding.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_RevertsWhenNetFl` `TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_RevertsWhenFeeRisesBelowMinNet`, `TempoFeeInclusiveWrap-perTest.testFuzz_SendOFTFeeInclusive_FloorAboveNetAlwaysReverts`)
- FeeManager lookup or write failure aborts before funding.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_FeeManagerReject`
- Balance failures or ending-below-baseline underflow revert.
  - ☐ Negative test
- A gross pull with allowance one unit below the requested amount reverts with the token's insufficient-allowance error.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_SingleApprovalOf`
- False-returning and malformed-return gross pulls remain untested.
  - ☐ Negative test
- Temporary allowance failure reverts.
  - ☐ Negative test

- Hop rechecks rely on trusted TIP-20 true-or-revert behavior.
  - ☐ Negative test
- Local transfer failures revert; unchecked low-160-bit recipient decoding permits a zero-equivalent local destination.
  - ☐ Negative test
- Remote hop failure rolls back all funding, and quoted-fee headroom caps the hop allowance so over-collection reverts.
  - ☑ Negative test (`TempoFeeInclusiveWrapperTest.test_SendOFTFeeInclusive_HopRevertRollsBa` `TempoFeeInclusiveWrap-perTest.test_SendOFTFeeInclusive_RevertsWhenHopCollectsMoreThanQuoted`)
- Other remote failures and bounded dust headroom behavior remain untested.
  - ☐ Negative test
- Allowance-cleanup failure rolls back source-side effects.
  - ☐ Negative test
- Refund or residual-accounting failure reverts.
  - ☐ Negative test
- Zero-equivalent recipients remain accepted, and the NatSpec claim is unenforced.
  - ☐ Negative test
- Fixture execution stopped at `PolicyForbids()` without application results.
  - ☐ Negative test

Function call analysis

- ASI trace: `this._previewFeeInclusive(...) -> this._requireHopAccepts(...) -> this.HOP.paused()/approvedOft(_oft)`.
- ASI trace: `IOFT(_oft).token() -> this.HOP.quoteStatic(...) -> this.HOP.removeDust(...)`.
- ASI trace: `TIP_FEE_MANAGER.userTokens(...) -> TIP_FEE_MANAGER.setUserToken(feeToken)`.
- ASI trace: `balanceOf -> safeTransferFrom -> forceApprove -> HOP.sendOFT -> forceApprove(0) -> balanceOf -> safeTransfer`.
- External edges trust hop/OFT/token/FeeManager/DEX behavior; the function accepts no native value, endpoint-dollar excess remains with the hop, and only token residuals return through the wrapper.
- **Manual call-path verification**: The `this.*` and `address(this.HOP)` notations are trace errors; review covered asynchronous delivery and the separate inbound replay-identity and queued-settlement custody behavior in `lzCompose`.

5.36. Module: TempoGasTokenBase.sol

Function: `_collectNativeAltToken(uint256 _nativeFee, address userToken, uint256 _maxUserTokenAmount)`

This function performs the core fee collection: it pulls `userToken` from `msg.sender`, swaps if needed, and holds the resulting whitelisted token for downstream payment.

Inputs

- `_nativeFee`
  - **Control**: Propagated from the endpoint fee; set by the protocol, not the caller.
  - **Constraints**: Must fit in `uint128` on the swap path.
  - **Impact**: Exact amount of the whitelisted target token acquired and held by this contract.
- `userToken`
  - **Control**: Resolved from `feeTokenOf(msg.sender)` or passed explicitly by the inheriting contract.
  - **Constraints**: No zero-address check or contract-existence check; a noncontract or zero address causes the downstream `safeTransferFrom` to revert.
  - **Impact**: The ERC-20 transferred from `msg.sender` to fund the fee — also determines which collection path (direct or swap) is taken.
- `_maxUserTokenAmount`
  - **Control**: Set by the caller (either `type(uint256).max` from the uncapped overload, or a user-supplied ceiling).
  - **Constraints**: Must be greater than or equal to the required fee.
  - **Impact**: Upper bound on what is debited from `msg.sender`; any excess pulled in the swap path is refunded after settlement.

Branches and code coverage

Intended branches

- Returns `address(0)` without pulling anything when `_nativeFee == 0`.
  - ☑ Test coverage (`RemoteMintRedeemHopTempoForkTest.testFork_ZeroFeeCollectsNothing`)
- Whitelisted path: transfers exactly `_nativeFee` of `userToken` from `msg.sender` and returns `userToken`.
  - ☑ Test coverage (`testFork_E2E_FrxUsd_PathUsdFee`)

- Swap path: settles for a whitelisted target within the padded input limit, clears the DEX allowance, leaves no input token on the hop, and binds and retains the returned target token.

  ☑ Test coverage (`testFork_E2E_FrxUsd_FrxUsdFeeViaDex`)

Negative behavior

- The base contract has no reentrancy guard. The swap callback window (`swapExactAmountOut`) executes while this contract holds `maxAmountIn` of `userToken` and a DEX allowance is live. Inheriting contracts are expected to carry the guard.

  ☐ Negative test
- Reverts with `FeeAboveCap` before any token pull when `_nativeFee > _maxUserTokenAmount` on the whitelisted path.

  ☑ Negative test (`testFork_E2E_FeeCap_WhitelistedToken`)
- Reverts with `FeeAboveCap` before any token pull when `maxAmountIn > _maxUserTokenAmount` on the swap path.

  ☑ Negative test (`testFork_E2E_FeeCap_DexRoutedToken`)

Function: `_collectNativeAltToken(uint256 _nativeFee)`

This function resolves the caller's fee token via `_resolveUserToken()` and delegates to the three-argument overload with `_maxUserTokenAmount = type(uint256).max`, imposing no spend cap.

Inputs

- `_nativeFee`
  - **Control**: Propagated from the endpoint fee.
  - **Constraints**: Must fit in `uint128` when the swap path is taken.
  - **Impact**: Amount of a whitelisted token collected by this contract to pay the endpoint.

Branches and code coverage

Intended branches

- Returns `address(0)` when `_nativeFee == 0`.

  ☐ Test coverage
- Collects `_nativeFee` of the caller's fee token.

  ☐ Test coverage — the previously mapped E2E path calls the three-argument

overload; no scoped source or test call reaches this uncapped wrapper

- The overload supplies `type(uint256).max`, so the caller has no explicit fee cap.
  - ☐ Test coverage — no scoped test assertion is mapped to the absence of a cap

Function: `_setFeeSwapSlippageBps(uint16 _bps)`

This function sets the slippage headroom applied to every future fee-swap quote. It is only callable by inheriting contracts.

Inputs

- `_bps`
  - **Control**: Controlled by the inheriting contract's admin path.
  - **Constraints**: Must be ≤ `MAX_FEE_SWAP_SLIPPAGE_BPS`.
  - **Impact**: Stores the new slippage allowance.

Branches and code coverage

Intended branches

- The internal setter stores the inclusive maximum and emits `FeeSwapSlippageBpsSet(200)`.
  - ☑ Test coverage (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`
- Passing 0 causes `feeSwapSlippageBps()` to return the default of 50 bps.
  - ☑ Test coverage (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`

Negative behavior

- The function reverts with `FeeSwapSlippageTooHigh` when `_bps > 200`.
  - ☑ Negative test (`RemoteMintRedeemHopTempoForkTest.testFork_FeeSwapSlippageBpsDefaultAnd`

Function: `_withSlippage(uint128 _amountIn)`

This function adds a configurable basis-point headroom to a DEX-quoted input amount, plus at least one unit.

Inputs

- `_amountIn`
  - **Control**: Propagated from the DEX quote returned by `_findSwapTarget`.
  - **Constraints**: Any `uint128` value — the function handles 0 and `type(uint128).max` explicitly.
  - **Impact**: Returns the padded amount used as both the cap passed to `swapExactAmountOut` and the amount pulled from `msg.sender`.

Branches and code coverage

Intended branches

- For ordinary nonzero DEX quotes, the returned maximum equals the raw quote padded by the configured basis points.
  - ☑ Test coverage (`RemoteMintRedeemHopTempoE2EForkTest.testFork_E2E_FrxUsd_FrxUsdFeeViaDe` `RemoteMintRedeemHopTempoE2EForkTest.testFork_E2E_FeeCap_DexRoutedToken`)
- Overflow and clamping boundaries remain untested.
  - ☐ Test coverage
- Returns `_amountIn + 1` when the percentage rounding produces no growth (tiny amounts or zero bps).
  - ☐ Test coverage
- Caps at `type(uint128).max` when the padded result overflows `uint128`.
  - ☐ Test coverage

Function: `feeTokenOf(address _caller)`

This function returns the TIP-20 token that will be debited when `_caller` sends without naming a fee token explicitly.

Inputs

- `_caller`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Address whose FeeManager `userTokens` slot is read.

Branches and code coverage

Intended branches

- Returns the address stored in `TIP_FEE_MANAGER.userTokens(_caller)` when it is nonzero.
  - ☑ Test coverage (`testFork_E2E_FeeTokenOfView`)
- Returns `PATH_USD_ADDRESS` when `userTokens(_caller)` is `address(0)`.
  - ☑ Test coverage (`testFork_E2E_FeeTokenOfView`)

Function: `quoteUserTokenFee(address _userToken, uint256 _endpointFee)`

This function estimates the amount of `_userToken` required to cover `_endpointFee`. UIs call this after `quoteSend()` to preapprove the correct amount.

Inputs

- `_userToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The token for which the fee estimate is returned.
- `_endpointFee`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must fit in `uint128` when the swap path is taken (`SafeCast.toUint128` reverts on overflow).
  - **Impact**: The endpoint-native fee amount to be expressed in `_userToken` units, including slippage.

Branches and code coverage

Intended branches

- Returns 0 when `_endpointFee == 0`.
  - ☐ Test coverage
- Returns `_endpointFee` unchanged (1:1) when `_userToken` is a whitelisted token.
  - ☑ Test coverage (`testFork_E2E_FrxUsd_PathUsdFee`)
- Returns quoted DEX input plus slippage allowance for a nonwhitelisted token.
  - ☑ Test coverage (`testFork_E2E_FrxUsd_FrxUsdFeeViaDex`)

Negative behavior

- Passing `address(0)` for `_userToken` silently substitutes `PATH_USD_ADDRESS` without informing the caller.
    - ☐ Negative test

5.37. Module: WFRAXTokenOFTUpgradeable.sol

Function: `checkpointRateLimits(uint32[] _eids)`

This owner-only function checkpoints each EID occurrence by selecting its override or default configuration, applying timestamp-based decay in both directions, writing to namespaced storage, and emitting `RateLimitCheckpointed`. Empty arrays succeed. Duplicate EIDs cause repeated writes and emissions but no additional same-timestamp decay.

Inputs

- `_eids`
    - **Control**: The owner controls array length, order, duplicates, and every `uint32` EID.
    - **Constraints**: Each EID selects its override or default configuration, stored state, and event key, with practical length bounded by gas.
    - **Impact**: Every occurrence updates `states[eid]` and emits `RateLimitCheckpointed`, including repeated or unconfigured EIDs.
- `msg.sender`
    - **Control**: The immediate caller determines `msg.sender`.
    - **Constraints**: The `onlyOwner` modifier requires the configured owner, and delegatecall preserves the caller.
    - **Impact**: It controls authorization but is not otherwise consumed or recorded.
- `msg.value`
    - **Control**: A caller can attempt to attach native value.
    - **Constraints**: The nonpayable entry point rejects any nonzero value before execution.
    - **Impact**: No native value is accepted, forwarded, refunded, or retained.
- `tx.origin`
    - **Control**: The transaction originator determines `tx.origin`.
    - **Constraints**: The wrapper, module, and library do not read it.
    - **Impact**: It has no authorization, storage, arithmetic, event, or interaction effect.
- `block.timestamp` and existing rate-limit storage
    - **Control**: The block producer supplies the timestamp, while prior operations

determine configuration and state.
- **Constraints**: The timestamp is narrowed to `uint32`, and checkpointing ignores global and directional enable flags.
- **Impact**: These values determine decay, retained usage, and the state and timestamp written and emitted.

Branches and code coverage

Intended branches

- Authorized wrapper path: the configured owner passes `onlyOwner` and reaches internal checkpoint dispatch.
  - ☐ Test coverage.
- Empty-array path: zero iterations return without writes or events.
  - ☐ Test coverage.
- Nonempty-array loop path: each occurrence is processed sequentially, written, and emitted.
  - ☐ Test coverage.
- Per-EID override path: an enabled override supplies both directions' limits and windows.
  - ☐ Test coverage.
- Default-config path: absent an override, the default configuration supplies both directions' parameters.
  - ☐ Test coverage.
- Initial or non-older timestamp path: usage is retained, and `lastUpdated` becomes `uint32(block.timestamp)`.
  - ☐ Test coverage.
- Older timestamp path: elapsed time drives independent outbound and inbound decay before advancing the shared timestamp.
  - ☐ Test coverage.
- No-decay-input path: zero usage, limit, or window retains that direction's usage.
  - ☐ Test coverage.
- Full-window path: elapsed time at least equal to the window reduces qualifying usage to zero.
  - ☐ Test coverage.
- Partial-window path: floored proportional replenishment either clears or reduces usage.
  - ☐ Test coverage.

- Repeated-EID path: later same-transaction occurrences retain usage but still write and emit.
  - ☐ Test coverage.
- Globally disabled or direction-disabled path: enable flags are ignored, so zero fields retain usage while nonzero fields still decay.
  - ☐ Test coverage.
- Checkpoint-before-default-reconfiguration path: checkpointing preserves listed EIDs' accrued recovery under the old effective slope before default replacement.
  - ☐ Test coverage.
- Shared-timestamp rounding path: both directions share a timestamp, so floored division can discard subunit recovery.
  - ☐ Test coverage.

Negative behavior

- Unauthorized caller: the `onlyOwner` modifier reverts before dispatch, leaving state and logs unchanged.
  - ☐ Negative test.
- Native-value submission: nonzero `msg.value` reverts before the function body.
  - ☐ Negative test.
- ABI-invalid input: malformed array encoding or noncanonical `uint32` values revert during decoding.
  - ☐ Negative test.
- Arbitrary EID is not rejected: zero, unconfigured, and duplicate EIDs remain valid inputs.
  - ☐ Negative test.
- An array whose calldata or loop cost exceeds available gas reverts and rolls back prior writes and logs.
  - ☐ Negative test.
- Linked-library failure: a delegatecall revert or out-of-gas failure bubbles up and rolls back the entire transaction.
  - ☐ Negative test.

Function call analysis

- `this._checkpointRateLimits(_eids) -> RateLimiterLib.checkpointRateLimits(_eids)`
  - The `this.` prefix is inaccurate. Internal module dispatch reaches the fixed linked library by `DELEGATECALL`, preserving caller context and using the

calling contract's storage. The reviewed library makes no nested external calls and returns no value.

- **Manual call-path verification**: The call path is `WFRAXTokenOFTUpgradeable.checkpointRateLimits` → internal `RateLimiterModule._checkpointRateLimits` → linked `RateLimiterLib.checkpointRateLimits` → effective-config selection, decay, namespaced state write, and `RateLimitCheckpointed`. The matching deployed library runtime remains a deployment assumption.

Function: `constructor(address _lzEndpoint)`

Deployment forwards `_lzEndpoint` unchanged to `OFTUpgradeable(address)` and then calls `_disableInitializers()` to prevent direct implementation initialization while proxy storage remains separate. Endpoint correctness is a deployment assumption because imported base and initializer implementations are unavailable.

Inputs

- `_lzEndpoint`
  - **Control**: The deployment transaction or creating factory selects `_lzEndpoint`, which is forwarded unchanged to the base constructor.
  - **Constraints**: The target only enforces the `address` ABI type and `msg.value == 0`; imported base acceptance rules are unknown.
  - **Impact**: The value may configure or be rejected by the inherited OFT constructor without target-local conversion, transfer, or event emission.

Branches and code coverage

Intended branches

- Inherited-constructor success: successful `OFTUpgradeable(_lzEndpoint)` construction reaches the target constructor body.
  - ☐ Test coverage
- Implementation-lock call: after base construction, the target calls `_disableInitializers()`.
  - ☐ Test coverage
- Successful creation: if both calls complete, deployment finishes without further target-local statements, events, transfers, or return values.
  - ☐ Test coverage

Negative behavior

- Nonpayable creation: creation with `msg.value != 0` reverts and deploys no contract.
  - ☐ Negative test
- Inherited-constructor failure: a base-constructor revert skips `_disableInitializers()`, rolls back construction, and deploys no contract.
  - ☐ Negative test
- Initializer-lock failure: a `_disableInitializers()` revert atomically rolls back all construction effects.
  - ☐ Negative test
- No target-local endpoint rejection: the target does not reject zero, EOA, unintended, or adversarial endpoint addresses.
  - ☐ Negative test

Function call analysis

- `constructor(address)` → `OFTUpgradeable(address)`
  - Explicit base-constructor chaining forwards `_lzEndpoint` unchanged in the creation context; no return value is consumed, and imported validation, state effects, events, nested interactions, and revert selectors cannot be established.
- `constructor(address)` → `_disableInitializers()`
  - This argumentless internal inherited call operates on construction storage to lock implementation initialization; any revert aborts deployment, while exact mutations and events depend on the unavailable helper implementation.
- **Manual call-path verification**: Manual tracing confirms only the explicit `OFTUpgradeable(_lzEndpoint)` base chain followed by `_disableInitializers()`; sibling constructors and the unrelated Tempo `nativeToken()` path are not reachable.

Function: `initializeV120()`

Any caller may execute this nonpayable `reinitializer(4)` migration while proxy storage remains eligible, setting fixed EIP-712 domain strings and clearing cached hashes. Success records version `4` and emits `Initialized(4)`; implementation initializers are constructor-disabled.

Inputs

- No parameters exist; the sender controls timing, `tx.origin` is unused, `msg.value` must be zero, and initialization storage controls admission.
- The `name()` and `version()` functions return `"Wrapped Frax"` and `"1.2.0"`; the fixed EIP-712 slot is caller-independent, while proxy eligibility depends on deployment and

upgrade state.

Branches and code coverage

Intended branches

- Eligible migration: when initialization is inactive and the stored version is below 4, the modifier enters initialization and executes the migration.
  - ☐ Test coverage
- Target name and version resolution: internal calls resolve the fixed strings `"Wrapped Frax"` and `"1.2.0"` without caller-controlled or mutable metadata.
  - ☐ Test coverage
- EIP-712 storage update: the helper stores both strings and zeroes `_hashedName` and `_hashedVersion` in the fixed namespace.
  - ☐ Test coverage
- Successful finalization: after successful writes, the modifier exits initialization, retains version 4, and emits `Initialized(4)`.
  - ☐ Test coverage
- Source-review coverage assessment: the body has no conditional statement, with admission and failure behavior arising from the dispatcher and `reinitializer(4)`.
  - ☐ Test coverage

Negative behavior

- Initialization already active: the `reinitializer(4)` modifier rejects execution from an active initialization context before the body runs.
  - ☐ Negative test
- Version already consumed: a stored initialization version of 4 or higher rejects replay before any EIP-712 write.
  - ☐ Negative test
- Direct implementation call: constructor-disabled implementation storage causes direct calls to fail the version gate without determining proxy state.
  - ☐ Negative test
- Nonzero native value: the nonpayable dispatcher rejects nonzero `msg.value` before executing the body.
  - ☐ Negative test
- Unrestricted caller identity: any caller may trigger the deterministic migration while eligible but cannot choose its strings, slot, or version.
  - ☐ Negative test

- Internal failure and rollback: any internal revert atomically restores initialization and EIP-712 state and suppresses `Initialized(4)`.
  - ☐ Negative test
- No interaction-driven behavior: the path performs no external call, callback, token movement, native transfer, bridge action, or reentrant interaction.
  - ☐ Negative test

Function call analysis

- `this.__EIP712_init(this.name(), this.version()) -> this.__EIP712_init_unchained(name, version) -> this._getEIP712Storage()`
- **Manual call-path verification**: The generated `this.` qualifiers are inaccurate because dispatch is internal; `_getEIP712Storage()` only assigns a fixed storage pointer, and ABI-absent `__EIP712_init` is reachable here only inside `reinitializer(4)`, not as an unguarded external initializer.

Function: `setDefaultRateLimitConfig(RateLimitConfig _defaultConfig)`

This owner-only setter validates and replaces namespaced defaults and emits `DefaultRateLimitConfigSet`. Default-backed EIDs later use them for burn/mint admission, decay, and `quoteOFT`; this call moves no value.

Inputs

- `_defaultConfig`
  - **Control**: The owner supplies all seven fields, while `msg.value` must be zero and the linked library address is not calldata controlled.
  - **Constraints**: Each enabled direction requires a nonzero limit and window; disabled placeholders are allowed, and the default's override flag does not select EID overrides.
  - **Impact**: The complete struct replaces namespaced defaults, while the event emits six fields excluding `overrideDefaultConfig` and existing usage remains uncheckpointed.

Branches and code coverage

Intended branches

- Authorized valid update: a valid owner call stores all seven fields, emits `DefaultRateLimitConfigSet`, and returns no data.
  - ☐ Test coverage

- Disabled-direction placeholders: a disabled direction may retain zero or nonzero limit and window values.
  - ☐ Test coverage
- Uncheckpointed default-backed EID: replacing defaults without checkpointing produced 1,000 available units in the supplied 1/1 reproduction.
  - ☐ Test coverage — audit-only reproduction, no scoped-repository test case was mapped
- Checkpointed default-backed EID: checkpointing under the old defaults before replacement produced 910 available units in the same reproduction.
  - ☐ Test coverage — audit-only reproduction, no scoped-repository test case was mapped
- Empty-return completion: successful validation, storage assignment, and event emission complete with empty return data.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a non-owner reverts before the helper call, leaving no state or logs.
  - ☐ Negative test
- Invalid outbound configuration: enabling outbound limits with a zero limit or window reverts before storage assignment.
  - ☐ Negative test
- Invalid inbound configuration: enabling inbound limits with a zero limit or window reverts before storage assignment.
  - ☐ Negative test
- Native-value attachment: the nonpayable entry point rejects nonzero `msg.value`.
  - ☐ Negative test
- Malformed calldata: invalid tuple encoding fails ABI decoding before the function body.
  - ☐ Negative test
- Library execution failure: any delegatecall failure reverts atomically, rolling back storage and logs.
  - ☐ Negative test

Function call analysis

- `this._setDefaultRateLimitConfig(_defaultConfig) -> RateLimiterLib.setDefaultRateLimitConfig(_defaultConfig)`

    - ASI renders an external self-call that does not occur. The source performs internal module dispatch before a deployment-linked library delegatecall that validates input, writes active-contract storage, and emits the event without callbacks or transfers.
- **Manual call-path verification**: The resolved path is `setDefaultRateLimitConfig` → internal `RateLimiterModule._setDefaultRateLimitConfig` → linked `RateLimiterLib.setDefaultRateLimitConfig` by delegatecall → validation → namespaced storage assignment → event emission.

Findings

- **Configuration-transition hazard**: Replacing defaults without checkpointing reinterprets existing default-backed usage under the new decay slope; the observed `1,000` versus `910` availability demonstrates an owner-controlled configuration-transition hazard, not an authorization bypass.

Function: `setRateLimitConfigs(SetRateLimitConfigParam[] _params)`

This owner-only function atomically checkpoints each EID in the batch under its prior effective config then stores and emits the replacements. Empty batches succeed, while duplicate EIDs are processed sequentially.

Inputs

- `_params`
    - **Control**: The owner controls array length, order, duplicate EIDs, and all config fields, while the environment supplies prior state, defaults, gas, linked code, proxy context, and `block.timestamp`.
    - **Constraints**: The call is nonpayable and owner-gated, enabled directions require nonzero limit/window pairs, disabled placeholders are unrestricted, and non-overrides still validate before using defaults.
    - **Impact**: Each element checkpoints and emits old effective state before storing and emitting its replacement, with capacity shared across users for each EID and timestamps truncated to `uint32`.

Branches and code coverage

Intended branches

- Authorized empty batch: the owner may submit an empty array, producing no per-EID writes or events.
    - ☐ Test coverage

- Authorized valid element: each valid element checkpoints under its old effective config, stores the replacement, and emits both events in order.
  - ☐ Test coverage
- Default-selected stored config: with `overrideDefaultConfig == false`, the supplied struct is stored and emitted, but subsequent effective reads use the default.
  - ☐ Test coverage
- Per-EID override: with `overrideDefaultConfig == true`, the stored values become effective for that EID.
  - ☐ Test coverage
- Disabled direction with zero placeholders: a zero disabled limit or window preserves usage while advancing the shared timestamp.
  - ☐ Test coverage
- Disabled direction with nonzero inactive fields: nonzero disabled placeholders can continue decaying historical usage.
  - ☐ Test coverage
- Initial or nonincreasing truncated timestamp: a zero or nonincreasing `uint32` timestamp retains usage and updates `lastUpdated`.
  - ☐ Test coverage
- Elapsed-time decay: decay clears, preserves, or reduces each direction using floored integer replenishment, causing fractional replenishment loss under the shared timestamp.
  - ☐ Test coverage
- Multiple or duplicate EIDs: elements execute sequentially, duplicate EIDs use the preceding replacement, and the batch commits atomically.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: a caller other than the current owner reverts before entering the helper.
  - ☐ Negative test
- Nonzero native value: the nonpayable guard rejects nonzero `msg.value`.
  - ☐ Negative test
- Malformed or noncanonical calldata: ABI decoding failures revert before batch processing.
  - ☐ Negative test
- Invalid outbound config: an enabled outbound direction with a zero limit or window reverts with `InvalidRateLimitConfig`.

- ☐ Negative test

- Invalid inbound config: an enabled inbound direction with a zero limit or window reverts with `InvalidRateLimitConfig`.
  - ☐ Negative test
- Failure after earlier elements: any later validation, library, or gas failure rolls back all prior writes and logs.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitConfigs(_params) -> RateLimiterLib.setRateLimitConfigs(_params)`
  - The source makes an internal helper call followed by a fixed-library `DELEGATECALL` that preserves sender/value, operates on proxy namespaced storage, returns no value, performs no callbacks or asset movement, and propagates failure.
- **Manual call-path verification**: The path is `WFRAXTokenOFTUpgradeable.setRateLimitConfigs` → `onlyOwner` → `RateLimiterModule._setRateLimitConfigs` → `RateLimiterLib.setRateLimitConfigs`. ASI renders the helper as `this.*` and omits authorization, delegatecall context, ordering, rollback, and the year-2106 timestamp assumption.

Function: `setRateLimitGlobalConfig(RateLimitGlobalConfig _globalConfig)`

This owner-only setter delegatecalls the linked RateLimiterLib in proxy context to store the namespaced boolean and emit `RateLimitGlobalConfigSet`. It leaves default/per-EID configurations and raw usage unchanged.

Inputs

- `_globalConfig`
  - **Control**: The caller supplies `isGloballyDisabled`, which is forwarded unchanged as the stored value and event argument after `msg.sender` passes `onlyOwner`.
  - **Constraints**: Calldata must contain a canonical boolean, `msg.sender` must be the configured owner, and successful calls require zero `msg.value`.
  - **Impact**: Setting `true` bypasses later rate-limit enforcement, while setting `false` restores it, without changing existing configurations or usage records.

Branches and code coverage

Intended branches

- Authorized enablement of enforcement: an authorized `false` stores and emits `false`, restoring effective default or per-EID enforcement.
  - ☐ Test coverage
- Authorized global disablement: an authorized `true` stores and emits `true`, causing consumption to skip usage updates and availability views to return `type(uint256).max`.
  - ☐ Test coverage
- Successful linked-library completion: the fixed linked-library `DELEGATECALL` preserves the storage write and event, and then the wrapper returns no data.
  - ☐ Test coverage
- No parameter-dependent branch inside the setter: the library stores and emits either valid decoded boolean.
  - ☐ Test coverage

Negative behavior

- Unauthorized caller: failure of `onlyOwner` reverts before the helper call, leaving no write or event.
  - ☐ Negative test
- Native-value call: nonzero `msg.value` reverts because the function is nonpayable.
  - ☐ Negative test
- Malformed calldata: insufficient data or a noncanonical boolean encoding reverts during ABI decoding.
  - ☐ Negative test
- Library-call failure: any delegatecall revert or exceptional halt propagates and atomically rolls back the write and event.
  - ☐ Negative test

Function call analysis

- `this._setRateLimitGlobalConfig(_globalConfig) -> RateLimiterLib.setRateLimitGlobalConfig(_globalConfig)`
  - The generated `this.` hop is a misrender: the wrapper calls the helper internally, which performs a fixed-destination `DELEGATECALL` preserving proxy address, storage, and original `msg.sender`; the library writes slot `0xf02f50e1c246a23e457c592e128a403f9c1cc60baa3a1ad2dc69bb4a5a51be00`,

emits the event, returns no value, and makes no callback or nested external call.

- **Manual call-path verification**: The verified path is external nonpayable dispatch → `onlyOwner` → internal `_setRateLimitGlobalConfig` → linked-library `DELEGATECALL` → namespaced write and event → empty return; ASI failures `FraxOFTMintableAdapterUpgradeableTIP20.sol@174:7143` and `FraxOFTUpgradeableTempo.sol@48:2180` concern sibling Tempo send/fee paths, not this declaration.

Function: `setRateLimitStates(SetRateLimitStateParam[] _params)`

This owner-only, nonpayable setter forwards `_params` unchanged to linked library code that validates and sequentially replaces fixed-namespace per-EID states. It moves no tokens or native value and returns no data.

Inputs

- `_params`
  - **Control**: The owner controls array length, order, EIDs, usages, and timestamps; empty arrays succeed, and repeated EIDs leave the final state.
  - **Constraints**: ABI decoding, zero value, and ownership are required; timestamps cannot be future, and zero timestamps require both usages to be zero.
  - **Impact**: Each valid element replaces `states[eid]` and emits matching ordered data, while any failure atomically removes all writes and logs.

Branches and code coverage

Intended branches

- An authorized empty array performs no writes or events and succeeds.
  - ☐ Test coverage — the supplied evidence reports `RateLimiterModuleTest` as 6/6 passing but does not identify an exact empty-array test
- One or more valid elements are validated, stored, and emitted in array order.
  - ☐ Test coverage — no named test or assertion mapping this complete success path is supplied
- An all-zero state with `lastUpdated == 0` is valid.
  - ☐ Test coverage — no exact all-zero-state test is identified
- Any nonzero, nonfuture timestamp is valid even when usage exceeds the effective limit.
  - ☐ Test coverage — no exact past/current-timestamp or over-limit usage test is

identified

- Duplicate EIDs emit each occurrence and leave the final occurrence stored.
  - ☐ Test coverage — no exact duplicate-EID ordering and event test is identified
- Both usages share one EID and `lastUpdated`; floored recovery can discard fractions, while disabled enforcement may retain yet ignore stored usage.
  - ☐ Test coverage — no exact test of this setter's downstream shared-state effects was identified

Negative behavior

- An unauthorized caller reverts before any library effect.
  - ☐ Negative test — no exact unauthorized `setRateLimitStates` test is identified
- A future `lastUpdated` reverts with `InvalidRateLimitState()`.
  - ☐ Negative test — no exact future-timestamp test is identified
- A zero timestamp with either usage nonzero reverts with `InvalidRateLimitState()`.
  - ☐ Negative test — no exact zero-timestamp/nonzero-usage test is identified
- Any invalid later element reverts the complete batch, including earlier writes and logs.
  - ☐ Negative test — no exact mixed valid/invalid batch rollback test is identified
- Malformed or out-of-range ABI calldata reverts during external dispatch.
  - ☐ Negative test — no exact malformed-calldata test is identified
- Nonzero `msg.value` reverts because the entry point is nonpayable.
  - ☐ Negative test — no exact nonpayable-value test is identified
- Insufficient gas during decoding, iteration, writes, or events reverts the complete transaction.
  - ☐ Negative test — no exact gas-boundary test is identified

Function call analysis

- `this._setRateLimitStates(_params) -> RateLimiterLib.setRateLimitStates(_params)`
  - The generated `this._setRateLimitStates` edge is source-inaccurate; the nonvirtual helper is called internally and then invokes the fixed-linked library by `DELEGATECALL`, preserving caller and storage context without callbacks, nested calls, return data, or caller-selected targets.
- **Manual call-path verification**: Dispatch enforces ABI shape, nonpayability, and ownership before `WFRAXTokenOFTUpgradeable.setRateLimitStates` → internal `RateLimiterModule._setRateLimitStates` → linked `RateLimiterLib.setRateLimitStates`; library failure propagates atomically, and linkage integrity remains trusted.

5.38. Module: FrxUSDCustodian.sol (in dev-staked-frax-erc4626)

Function: `_isFeeExempt()`

This function returns `true` if either `msg.sender` or `tx.origin` appears in the `feeExempt` mapping. All four `preview*` overrides gate on this.

Branches and code coverage

Intended branches

- Returns `true` when `msg.sender` is in `feeExempt`.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- Returns `true` when `tx.origin` is in `feeExempt` and `msg.sender` is not.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- Returns `false` when neither address is exempt.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)

Function: `previewDeposit(uint256 _assetsIn)`

This function overrides the base `previewDeposit`. It applies `mintFee` only when `_isFeeExempt()` returns `false`.

Inputs

- `_assetsIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No explicit zero check.
  - **Impact**: Determines how many shares are minted.

Branches and code coverage

Intended branches

- Fee-exempt caller: the fee is zero, and the full `_assetsIn` converts to shares.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- Nonexempt caller with nonzero `mintFee`: `_assetsIn` is reduced by the fee fraction before conversion.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_deposit`,

FrxUSDCustodianTest.test_feeExempt)

- With `mintFee == 0`, repeated deposit round trips avoid any fee loss beyond the asserted one-unit rounding tolerance.

  ☑ Test coverage (`FrxUSDCustodianWithOracleTest.test_noFeeRounding`)

Function: `previewMint(uint256 _sharesOut)`

This function overrides the base `previewMint`. It applies `mintFee` only when `_isFeeExempt()` returns `false`.

Inputs

- `_sharesOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Target number of shares — determines the assets that must be supplied.

Branches and code coverage

Intended branches

- Fee-exempt caller: No fee markup is applied to assets.

  ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- Nonexempt caller: `_assetsIn` follows the `1e18 / (1e18 - fee)` gross-up formula.

  ☑ Test coverage (`FrxUSDCustodianTest.test_mint`)

  - The implementation rounds up; the test's one-unit tolerance does not independently distinguish ceiling from floor rounding.

Function: `previewRedeem(uint256 _sharesIn)`

This function overrides the base `previewRedeem`. It applies `redeemFee` to reduce `_assetsOut` after converting shares to assets.

Inputs

- `_sharesIn`
  - **Control**: Fully controlled by the caller.

- **Constraints**: No zero check.
- **Impact**: Number of shares to redeem.

Branches and code coverage

Intended branches

- Fee-exempt caller: the full asset value is returned.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- Nonexempt caller: `_assetsOut` follows the `(1e18 - redeemFee) / 1e18` fee-reduction formula.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_redeem`)
    - The implementation rounds down; the test's exactly divisible input does not independently exercise a remainder.

Function: `previewWithdraw(uint256 _assetsOut)`

This function overrides the base `previewWithdraw`. It applies `redeemFee` to gross up `_assetsOut`.

Inputs

- `_assetsOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero check.
  - **Impact**: Desired net asset withdrawal — determines shares required.

Branches and code coverage

Intended branches

- Fee-exempt caller: no gross-up is applied, and shares are calculated directly from `_assetsOut`.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- Nonexempt caller: `_assetsOut` follows the `1e18 / (1e18 - redeemFee)` gross-up formula before share conversion.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_withdraw`)
    - The implementation rounds up; the test's one-unit tolerance does not independently distinguish ceiling from floor rounding.

Function: `setFeeExempt(address _user, bool _isExemptFromFee)`

This is an owner-only function that adds or removes an address from the `feeExempt` mapping.

Inputs

- `_user`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No zero-address check.
  - **Impact**: The address granted or revoked fee exemption.
- `_isExemptFromFee`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: Enables or disables the fee for that address across all deposit, mint, withdraw, and redeem flows.

Branches and code coverage

Intended branches

- Grants fee exemption in the mapping.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- The grant emits the fee-exemption event.
  - ☐ Test coverage — event emission is not asserted
- Revoking the direct exemption restores fee charging when neither `msg.sender` nor `tx.origin` is exempt.
  - ☑ Test coverage (`FrxUSDCustodianTest.test_feeExempt`)
- The revocation emits the fee-exemption event.
  - ☐ Test coverage — event emission is not asserted

Negative behavior

- Reverts when called by a non-owner.
  - ☑ Negative test (`FrxUSDCustodianTest.test_feeExempt`)

5.39. Module: FrxUSDCustodian.sol (in fraxnet-dev)

Function: `deposit(uint256 _assetsIn, address _receiver)`

This function accepts a specified amount of `custodianTkn` from `msg.sender` and mints the corresponding amount of frxUSD (after mint fee) to `_receiver`.

Inputs

- `_assetsIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxDeposit(_receiver)`.
  - **Impact**: Amount of `custodianTkn` taken from the caller.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero-address check.
  - **Impact**: Recipient of the minted frxUSD.

Branches and code coverage

Intended branches

- The caller's `custodianTkn` decreases by `_assetsIn`.
  - ☐ Test coverage
- The `frxUSDMinted` state variable is updated to reflect new shares.
  - ☐ Test coverage
- The `Deposit` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function reverts if `_assetsIn` exceeds `maxDeposit`.
  - ☐ Negative test

Function: `initialize(address _owner, uint256 _mintCap, uint256 _mintFee, uint256 _redeemFee)`

This function initializes the proxy instance with an owner and fee parameters. The constructor sets `wasInitialized = true` on the implementation, so `initialize` can only be called once on the proxy.

Inputs

- `_owner`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not be the zero address.
  - **Impact**: Becomes the owner with full privileged access over the contract.
- `_mintCap`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets `mintCap`.
- `_mintFee`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets `mintFee`.
- `_redeemFee`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Sets `redeemFee`.

Branches and code coverage

Intended branches

- The owner is set correctly, and the contract is marked initialized.
  - ☐ Test coverage
- Fees and the mint cap are set correctly.
  - ☐ Test coverage

Negative behavior

- The function reverts if already initialized.
  - ☐ Negative test
- The function reverts if `_owner == address(0)`.
  - ☐ Negative test
- The function does not revert if `_mintFee` or `_redeemFee` ≥ `1e18`.
  - ☐ Negative test

Function: `mint(uint256 _sharesOut, address _receiver)`

This function mints a specified number of frxUSD shares to `_receiver` by pulling the required amount of `custodianTkn` (including mint fee) from `msg.sender`.

Inputs

- `_sharesOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxMint(_receiver)`.
  - **Impact**: Exact number of frxUSD shares minted.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero-address check.
  - **Impact**: Recipient of the minted frxUSD.

Branches and code coverage

Intended branches

- The caller pays `previewMint(_sharesOut)` of `custodianTkn`; the receiver gets exactly `_sharesOut` of frxUSD.
  - ☐ Test coverage
- The `frxUSDMinted` state variable is updated.
  - ☐ Test coverage
- The `Deposit` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function reverts if `_sharesOut` exceeds `maxMint`.
  - ☐ Negative test

Function: `recoverERC20(address _tokenAddress, uint256 _tokenAmount)`

This function allows the owner to sweep any ERC-20 token from the contract to the owner address, with no exclusion list.

Inputs

- `_tokenAddress`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: No token exclusion list.
  - **Impact**: Determines which token is recovered from the contract.
- `_tokenAmount`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: Limited only by the contract's balance of that token.
  - **Impact**: How much is recovered.

Branches and code coverage

Intended branches

- Owner recovers an accidentally sent ERC-20 token.
  - ☐ Test coverage
- The `RecoveredERC20` event is emitted.
  - ☐ Test coverage

Function: `redeem(uint256 _sharesIn, address _receiver, address _owner)`

This function burns a specified amount of frxUSD from the caller and releases the corresponding `custodianTkn` (after redeem fee) to `_receiver`.

Inputs

- `_sharesIn`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxRedeem(_owner)`.
  - **Impact**: Exact number of frxUSD shares burned.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero-address check.
  - **Impact**: Recipient of the `custodianTkn`.
- `_owner`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal `msg.sender`.

- **Impact**: Source of the frxUSD to be burned.

Branches and code coverage

Intended branches

- The caller's frxUSD decreases by `_sharesIn`; the receiver gets `previewRedeem(_sharesIn)` of `custodianTkn`.
  - ☐ Test coverage
- The `frxUSDMinted` state variable is decremented.
  - ☐ Test coverage
- The `Withdraw` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function reverts if `_owner != msg.sender`.
  - ☐ Negative test
- The function reverts if `_sharesIn` exceeds `maxRedeem`.
  - ☐ Negative test
- The `frxUSDMinted` state variable is silently clamped to 0 rather than reverting when `_sharesIn > frxUSDMinted`.
  - ☐ Negative test

Function: `shuffleToRwa(uint256 amount, uint256 minAmountRwaOut, uint8 assetToShuffleTo)`

This function allows an approved operator to move excess `custodianTkn` into RWA-earning positions via hardcoded external contracts.

Inputs

- `amount`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: The remaining balance must satisfy `custodianTkn.balanceOf(address(this)) - amount >= minAfterShuffle`.
  - **Impact**: How much `custodianTkn` is moved out, directly reducing the assets backing frxUSD in this contract.

- `minAmountRwaOut`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Only enforced when `assetToShuffleTo == 0` and `minAmountRwaOut > 0`. Completely ignored when `assetToShuffleTo == 1`.
    - **Impact**: Intended slippage floor for the RWA subscription.
- `assetToShuffleTo`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: Routes funds to either the hardcoded `rwaIssuer` / `rwaCustodian` or `wtgxxCustodian`.

Branches and code coverage

Intended branches

- When `assetToShuffleTo == 0`, `custodianTkn` is approved and subscribed via `IRWAIssuer`.
    - ☐ Test coverage
- When `assetToShuffleTo == 1`, `custodianTkn` is transferred to `wtgxxCustodian` and `shuffleToWtgxx` is called.
    - ☐ Test coverage

Negative behavior

- The function reverts if `msg.sender` is not an approved operator.
    - ☐ Negative test
- The function reverts if `amount` would bring the balance below `minAfterShuffle`.
    - ☐ Negative test
- The function reverts if slippage is too high on the `assetToShuffleTo == 0` path.
    - ☐ Negative test
- The `minAmountRwaOut` parameter is silently ignored when `assetToShuffleTo == 1`.
    - ☐ Negative test
- Values of `assetToShuffleTo` greater than 1 (e.g., 2, 255) silently route to the WTGXX branch without revert.
    - ☐ Negative test

Function: `withdraw(uint256 _assetsOut, address _receiver, address _owner)`

This function burns the caller's frxUSD (computed from `_assetsOut` via `previewWithdraw`) and releases `_assetsOut` of `custodianTkn` to `_receiver`.

Inputs

- `_assetsOut`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not exceed `maxWithdraw(_owner)`.
  - **Impact**: Amount of `custodianTkn` received by `_receiver`.
- `_receiver`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: No zero-address check.
  - **Impact**: Recipient of the `custodianTkn`.
- `_owner`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must equal `msg.sender`; otherwise, the function reverts with `TokenOwnerShouldBeSender`.
  - **Impact**: Source of the frxUSD to be burned.

Branches and code coverage

Intended branches

- The caller's frxUSD decreases by `previewWithdraw(_assetsOut)`.
  - ☐ Test coverage
- The `frxUSDMinted` state variable is decremented.
  - ☐ Test coverage
- The `Withdraw` event is emitted.
  - ☐ Test coverage

Negative behavior

- The function reverts if `_owner != msg.sender`.
  - ☐ Negative test
- The function reverts if `_assetsOut` exceeds `maxWithdraw`.
  - ☐ Negative test

- The `frxUSDMinted` state variable is silently clamped to 0 rather than reverting when the burn amount exceeds the tracked minted supply.

  - ☐ Negative test

6. Assessment Results

During our assessment on the scoped frxUSD Ecosystem contracts, we discovered 20 findings. No critical issues were found. Seven findings were of low impact and the remaining findings were informational in nature.

Zellic also matched the deployed implementation contracts to the reviewed code, including the remediation commits referenced in this report. The addresses and their source commits are listed in the Deployed Implementations section (6.1. ↗).

6.1. Deployed implementations

After remediation, Zellic matched the deployed implementation contracts listed below to the reviewed source code. For each address, Zellic compiled the cited commit locally with the compiler version and settings used for the deployment and compared the output with the runtime bytecode on chain as of October 28th, 2026. Apart from immutable values set at construction, such as LayerZero endpoint and token addresses, the bytecode is identical for every address listed.

Each cited commit is the reviewed commit, with the remediation commits referenced in this report applied where they were deployed, and contains no other changes to the listed contracts. Deployments that do not include an available remediation commit are noted in the relevant subsection. This section covers implementation contracts only; it does not cover proxy configuration, upgrade state, or other on-chain parameters.

frxUSD and sfrxUSD

These contracts are deployed on Ethereum and built from `FraxFinance/frax-tokens` @ `f0f24fb59fb2ea234afc38e5bb45aeee54c76768` ↗.

Contract	Address
FrxUSD	`0x5c19D7C2838769ED3c5a1E506600940e62fF901F`
SfrxUSD	`0xAad4A1D92053a62cE7a787641d8b4E5883e96700`

Finding 3.4. ↗ was acknowledged and is not fixed in this implementation.

FraxswapRouterMultihopV2

These contracts are deployed on Ethereum. The router is built from FraxFinance/dev-fraxswap-v3 @ 488caac2271217cf6a5b817fb7e1fc4d61f8956d, and the executor is deployed by the router's constructor.

Contract	Address
FraxswapRouterMultihopV2	0x00000000580F8b07840ECbFA1c7fAA8D3e6434D7
FraxswapExecutorV2	0x3548Eb1185C482C0F3eCa3c280E767c2efAe8039
PancakeV3Plugin	0xdF1405328ad2E677F9D3d3E0bdfd843168567747

The PancakeV3Plugin was deployed from FraxFinance/dev-fraxswap-v3 @ cca72784197038d57d31dd0a6b5f23c32c265c95. The remediation commit changes only its comments, so the deployed plug-in is unaffected.

frxUSD custodians

These custodians are deployed on Ethereum.

Custodian	Address
frxUSD-USDB	0x82487E965158606f062D9224605D799e6D69B6cB
frxUSD-BUIDL	0x329BcB20B8155bdD2E0Ce47539CB96a303e35189
frxUSD-USTB	0x7209ed865526A6dc431e9045307325c192387F48
frxUSD-USDC	0x016cF1b38a860fEfa6cCfe7f70a229f702a70Aa2
frxUSD-WTGXX	0xceAdE8cd22ce4500142dc4a2dDe583397Fb5d847

The USDB and BUIDL custodians (FrxUSDCustodian) are built from FraxFinance/dev-staked-frax-erc4626 @ 67c54c486a73ae042abcd3b12cff04876d7f0d22.

The USTB custodian (FrxUSDCustodianWithOracle) is built from FraxFinance/dev-staked-frax-erc4626 @ 05dbeade266d24a8cda3794004a5aeba1025ec7c.

The USDC custodian (FrxUSDCustodianUsdcV2) and WTGXX custodian (FrxUSDCustodianWithReceiverV2) are built from FraxFinance/fraxnet-dev @ b1dc549110b1e50e3124f9638905d01a1d749c17.

The remediation for Finding 3.18. is not included in the USDB, BUIDL, USDC, and WTGXX

implementations; it only affects initialization, and Frax stated that all of these custodians are already initialized. The source verified on the block explorer for the USDC and WTGXX implementations differs from the cited commit in whitespace and function-modifier order only; the cited commit compiles to the deployed bytecode.

Tempo hops

These contracts are deployed on Tempo.

Contract	Address
RemoteHopV201Tempo	0x8c48927557F6c1D55E82f3145b730b3AEBf8C058
TempoFeeInclusiveWrapper	0x2cfcA4A7ecE48860bfb78B7BD7Bd8CC9Ef75DaA3
RemoteMintRedeemHopTempo	0xDCe111dBBc88BF5D826B4dF5105010EA77db73cB

RemoteHopV201Tempo and TempoFeeInclusiveWrapper are built from dhruvinparikh/hop-v2 @ 43f99bc4926aba594cef8c139e756e50f1b3400d ↗.

RemoteMintRedeemHopTempo is built from dhruvinparikh/fraxtal-lz-hop @ 018261ce60143116b1c7e25c037457949af9b5f1 ↗.

frxUSD policy administrator

This contract is deployed on Tempo and built from FraxFinance/frax-oft-upgradeable @ 981a7b96fd85cd0d6e32cd1e1fd46331fbe6aebe ↗.

Contract	Address
FrxUSDPolicyAdminTempo	0xB7E34f51eC7417e3E0Afb91bD5E33657d911A8d7

The remediation for Finding 3.9. ↗ is not included in this implementation; Frax stated that the deployed instance was initialized through `initialize()` with a valid self-administered BLACKLIST policy and will not be redeployed.

OFT v1.2.0

Source: FraxFinance/frax-oft-upgradeable @ 5ce22ca59ccd3ac1fb16ca684c67bd9882981441 ↗, with the LayerZero OApp fork at FraxFinance/LayerZero-v2-upgradeable @ 9876b375a3c3bf490b4c6a37dcb889441043b2b4 ↗.

Linked libraries

The libraries are deployed with `CREATE2` and share one address on all 24 EVM chains listed in this section. zkSync Era uses separate deployments.

Library	Address
EIP3009Lib	`0x1539635F803C448B7FBeeBe09dD94e17C33Fae8e`
EIP712Lib	`0x7eA2F53bbD489EBc0195f3EDB99AC4a87083FF55`
FreezeThawLib	`0x5f835cE8434c852A4063e419951143987650930A`
PauseLib	`0xB0140DBA52403AdA53A1aa2efb52fc185FcC2092`
PermitLib	`0x24b8D15113493aEd8f556Df3D4eCC3842D3B39f8`
RateLimiterLib	`0x3612e32988Ca4db94B562d6341ce28F1299393f0`
TempoAltTokenLib (Tempo only)	`0x2D6433D9d67d121c234AeA77Ae82effd616A737F`

Library (zkSync Era)	Address
EIP3009Lib	`0xA1c81d54C6294C9bdd40FD330b9329c759223A35`
EIP712Lib	`0x02FC8aB03D6B384b3402a264B404CfE5c24a5EB3`
FreezeThawLib	`0x426b8547C8Fb1DcC700B5B8303eB450F503C84e2`
PauseLib	`0xbfd913EBE281FfAF01076383424af1f4dEA02C93`
PermitLib	`0x71f153BF29629Fd82d778beB8109a8DBD8D94b18`
RateLimiterLib	`0x7755f7eBB73B65c7F8c69D70105b5b926BaB4d62`

The zkSync Era PauseLib was deployed on September 18th, 2026, and reused. Its library source is unchanged in the cited commit, and its bytecode matches apart from the embedded metadata hash, which reflects the earlier build configuration.

Implementations

Implementations are deployed with `CREATE`, so an address is specific to its chain, and the same address can hold a different contract on another chain. Each table lists the chains on which the address holds that contract.

WFRAXTokenOFTUpgradeable

Chain(s)	Address
Avalanche, BNB Chain, HyperEVM, Ink, Linea, Optimism, Plume, Unichain, World Chain	`0x795C6C74506eF5889CBDebc3b04b1FBeA0551644`
Arbitrum, Polygon, Sonic	`0xe70c5fdD87C69e06aCff6761928548c18e154834`
Base, Katana, X Layer	`0xdAaf922A398Bf17c335248c376C289B69dFa2095`
Celo	`0x4F62251FA593c61282017ab6FD79E9F6f87a803C`
Ethereum	`0x7D412dfC7EeF6B04D0af258830B203F05598A5a8`
Monad	`0xa9D8C167bf89be571A8c5125AE49fE2C0B3e58D5`
Robinhood Chain	`0xb14B7F37ff0b2a2adbF90a0409A8fBFb32a8Eca6`
Sei	`0x0B323d0459D630fa2BeA59dDA3236F5d6d109d98`
Somnia	`0xD39822e8ef895dcE3471a0a80e162cb8548FFD41`
Stable	`0x4818AAB499EC0F5d816968899Bf69ECD5c39F354`
zkSync Era	`0x34F1696B5c68Db6531D7c5b2eD2433486a1F2C1c`

SFrxUSDOFTUpgradeable

Chain(s)	Address
Avalanche, BNB Chain, HyperEVM, Ink, Linea, Optimism, Plume, Unichain, World Chain	0xe70c5fdD87C69e06aCff6761928548c18e154834
Arbitrum, Polygon, Sonic	0xdAaf922A398Bf17c335248c376C289B69dFa2095
Base, Katana, X Layer	0xE7bFf72F843CC945a656e158696eb983dc1b3A66
Celo	0xD2B51b417B184a1c0C197EC653FC9601b50F2FbB
Monad	0x9Ea657A9a40D398389F521De000A06Ae4E43b6d4
Robinhood Chain	0xa4dCb15E851cC4F14498cc9B0158c317672Df7b5
Sei	0x921241205a1032a53Ff1cad8800e7F3D5F8FB8e3
Somnia	0x1714FC5aA9736b05fd68bC658a7Ec3996d1d3ccd
Stable	0x795C6C74506eF5889CBDebc3b04b1FBeA0551644
zkSync Era	0xD2C8d8925d165b16eB27c7585A89fD977908d8Ae

FrxUSDOFTUpgradeable

Chain(s)	Address
Avalanche, BNB Chain, HyperEVM, Ink, Linea, Optimism, Plume, Unichain, World Chain	0xE7bFf72F843CC945a656e158696eb983dc1b3A66
Arbitrum, Polygon, Sonic	0x05a4a277866db17f20A118630FdE7C35e0da2112
Base, Katana, X Layer	0x7D412dfC7EeF6B04D0af258830B203F05598A5a8
Celo	0xe9F94d978E9aFA0E732c7595530645627C62f871
Monad	0x595D0B828EBD17a6493879dd45e74003c5b2DdD5
Robinhood Chain	0x05b4a311Aac6658C0FA1e0247Be898aae8a8581f
Sei	0xB05db828c8499a0eC42778422aAeA49C37C7A7b1
Somnia	0x2f029ff7e543b13389bf8b4d7f9997098b2a8326
Stable	0xdAaf922A398Bf17c335248c376C289B69dFa2095
zkSync Era	0x1231bca48C801DA6ED8cC417f3a6E97B5A2Dd380

FraxOFTUpgradeable

It is used by sfrxETH and frxETH.

Chain(s)	Address
Avalanche, BNB Chain, HyperEVM, Ink, Linea, Optimism, Plume, Unichain, World Chain	0xdAaf922A398Bf17c335248c376C289B69dFa2095
Arbitrum, Polygon, Sonic	0xE7bFf72F843CC945a656e158696eb983dc1b3A66
Base, Katana, X Layer	0x05a4a277866db17f20A118630FdE7C35e0da2112
Celo	0xc2632c0515efD2A2F9b2a939ab05A87785673465
Monad	0x95620302C0850593435e52BB027DdB0329190A53
Robinhood Chain	0xd45530D08CA58Eefe103495236D63cE69190fdBd
Sei	0xDCD2ee53fb631981AaD03B5B97d01b1BAE70F7B0
Somnia	0xA093CF2c332364b19BEBc3531168987e115b5748
Stable	0xe70c5fdD87C69e06aCff6761928548c18e154834
zkSync Era	0x10B7592D40fFe6A6fAe289e51889Fa1098b019F9

FraxOFTMintableAdapterUpgradeable

It is used by the frxUSD and sfrxUSD lockboxes.

Chain	Address
Ethereum (sfrxUSD)	0x7786c1d12F0e5eF1c1eC47DC00Faa846e6C84c68
Ethereum (frxUSD)	0x765d9CACf678e1A7c3530F84871CedD155608D75
Fraxtal (sfrxUSD)	0xc6D5dA387BaF20931688e9a1116683a97fC5aE0c
Fraxtal (frxUSD)	0xEE5CAD1F02CF1cb5507C363077D0238B5fe71232

FraxOFTAdapterUpgradeable

It is used by the WFRAX, sfrxETH, and frxETH lockboxes.

Chain	Address
Ethereum (sfrxETH)	0x2a959A46FBD2b16b8C82E28c9Cce13B764A71d4F
Ethereum (frxETH)	0xa9D8C167bf89be571A8c5125AE49fE2C0B3e58D5
Fraxtal (WFRAX)	0xB05db828c8499a0eC42778422aAeA49C37C7A7b1
Fraxtal (sfrxETH)	0x6F097aC5e580ccD80CC16c982A6b38041eAe2C76
Fraxtal (frxETH)	0x977Cd936099345e7a96C7364E68e2cd71eBaAE83

FraxOFTUpgradeableTempo

It is used by WFRAX, sfrxUSD, sfrxETH, and frxETH on Tempo.

Chain	Address
Tempo	0xc488281Cd33A177fBFd926DeF11FdEF64E3F600A

FraxOFTMintableAdapterUpgradeableTIP20

It is used by frxUSD on Tempo.

Chain	Address
Tempo	0xC587603AEfDe5c5001fF2feE5B32eF6a5549782f

6.2. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version

reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Where Zellic states that a finding has been addressed or fixed in one or more specific commits, this indicates only that those commits introduce changes that, in our assessment, address the finding relative to the codebase version originally reviewed. This does not imply that Zellic reviewed other changes contained in those commits, the overall state of the codebase at those commits, or the interplay between remediation measures for different findings.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.