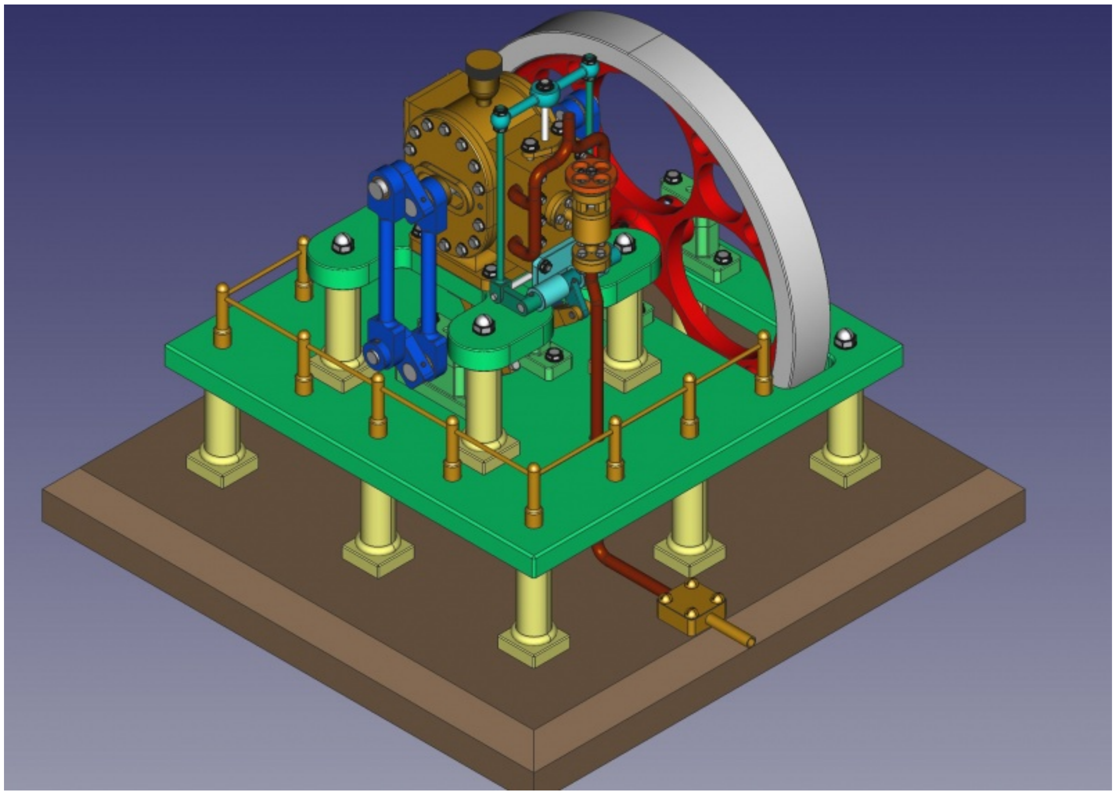
Module developer's guide to FreeCAD source code



for FreeCAD version 0.19-dev

Hosted https://github.com/qingfengxia/FreeCAD_Mod_Dev_Guide

by Qingfeng Xia 2016~2019

Module developer's guide to FreeCAD source code

by Qingfeng Xia 2016~

by Luzpaz 2019~

download the latest version from `pdf` folder of this repo

Changelog

- 2015-09-18 version 0.1 *for FreeCAD version 0.16-dev*
- 2016-09-18 version 0.2 *for FreeCAD version 0.17-dev*
- 2019-06-18 start again to work towards version 0.3 *for FreeCAD version 0.19-dev*

Plan and progress

This book should be updated for the recent release, esp. after migration to Python3 + Pyside2. I plan another release for FreeCAD 0.19 dev near Xmas time.

License of this book

Acknowledge to developers of FreeCAD

Original/lead developers:

- Jürgen Riegel
- Werner Mayer
- yorik van havre

Add all contributors see https://www.freecadweb.org/wiki/Contributors

Target audiences: new module developers

Make sure you are familiar with FreeCAD workbench GUI and API as a user:

- Foundamental document on official wiki for FreeCAD
- FreeCAD python API document
- single file PDF user manual for quick start

Doxygen documents links

Doxygen generated online documentation of source for 0.16dev, will be delete soon

Doxygen generated online documentation of source for 0.19dev on 2019-06-24

Why I want to write this book

- Learn the software architecture of FreeCAD: a large open source project
- Learn to use git to contribute to open source projects like FreeCAD
- Save time for new developers to explore the source code of FreeCAD
- Record personal note and lesson during writing/contributing code to FreeCAD
- Some chapters of this ebook is seeking to be merged into official wiki after reviewed as usable

My research: "Automated and Intelligent Engineering Design"

How to contribute to this ebook

- write on unfinished topics/chapters listed in todo.md fork and pull request
  `git clone https://github.com/qingfengxia/FreeCAD_Mod_Dev_Guide.git`
- file bug for outdated code analysis As this book is about code analysis while FreeCAD is under heavy development, source codes quoted may outdated quickly. Please point out my report bugs in this github issues board
- This ebook is pre-processed by a python script, see more details in the scripts folder's Readme.

There are some anchor texts like:

- [src/*.h/cpp] are processed into link to the latest official FreeCAD source.
- some another anchors "## folders . . . ", which will be marked out soon more explicitly

Acknowledgement to my family

This work is not funded to my employers (Oxford Unviersity, UKAEA) by the time 2019. It is a community voluntary work, thank every one review, contribute to this book.

Qingfeng Xia thanks for my wife Ms J. Wang, and other family members' for the housework exempt to complete this work.

Contents

Chapter 1

FreeCAD overview and architecture

First of all, thanks to the original developers (**Jürgen Riegel, Werner Mayer, Yorik van Havre**), for sharing this great artwork freely. FreeCAD is released under LGPL license, free for commercial usage with dynamic linkage.

1.1 Introduction to FreeCAD

The birth of FreeCAD: version 0.0.1 October 29, 2002 Initial Upload

wikipedia of FreeCAD

> FreeCAD is basically a collage of different powerful libraries, the most important being openCascade, for managing and constructing geometry, Coin3D to display that geometry, Qt to put all this in a nice Graphical User Interface, and Python to give full scripting/macro functions.

1.2 Key features

- Multiple platform: windows, Linux and MacOS X
- Console mode which can be imported by python module
- parametrized modelling, scripting and macro support, just like commercial CAD tool
- modular architecture with various plugins: CAD, CAM, robot, meshing, FEM, etc.
- supporting plenty of standard 2D and 3D CAD exchange file types, STL, STEP, etc.
- file type *.fcstd zip file container of many different types of information, such as geometry, scripts or thumbnail icons

example of embedding FreeCAD python module into Blender

see FreeCAD official website feature list for detailed and updated features

1.3 Software architecture

1.3.1 Key software libraries

see FreeCAD source code structure in Chapter 2

standing on giant's shoulder

- OpenCASCADE as CAD kernel
- OpenInventor/Coin3D/pivy for 3D scene rendering
- Qt and PySide for GUI
- Python scripting and wrapping: PyCXX, swig, boost.python
- Other powerful software libraries like Xerces XML, boost

The geometry algorithm using OpenCASCADE will be discussed in Chapter 7.

1.3.2 Mixed python and c++

- python scripting in console mode and python-based macro recording in GUI mode
- all FreeCAD class is derived from this *BaseClass*, connected with *BaseClassPy*
- c++11 is not extensively used before 0.17
- c++ template is not heavily used, but `FeatureT<>` make `DocumentObject, ViewProvider` extensible in Python
- FreeCAD not tied to Qt system until GUI, `Boost::signal` is used in command line mode: FreeCADCmd
- `std::string`(UTF8) is used internally, using `QString  getString(){QString.fromUtf8(s.c_str())}`
- C++ for most of time consuming task (threading model), to avoid bottleneck of Global Interpreter Lock

Mixing C++ and Python in module development will be discussed in Chapter 5.

1.3.3 GPL code will not be included into installer

https://github.com/yorikvanhavre/Draft-dxf-importer

Current FreeCAD policy is to include only LGPL software and no GPL by default. Mentioned DXF import-export libraries were downloaded by default. On DXF import-export operation in the past but Debian didn't like that and FreeCAD changed in a way user has to manually enable (Opt-In) the download.

Open **Draft** workbench and after that select *Edit -> Preferences*. Under *Import-Export -> DXF / DWG* tab, enable *Automatic update*. After that FreeCAD will download mentioned libraries on first DXF import-export operation and it should work. If it does not work restart FreeCAD and try again.

1.4 How 3D model are renderred

1.4.1 OpenInventor in FreeCAD's ViewProvider

The geometry that appears in the 3D views of FreeCAD are rendered by the Coin3D library. Coin3D is an implementation of the OpenInventor standard, which exempt you from OpenGL coding.

FreeCAD itself features several tools to see or modify openInventor code. For example, the following Python code will show the openInventor representation of a selected object:

```
obj = FreeCAD.ActiveDocument.ActiveObject
viewprovider = obj.ViewObject
print viewprovider.toString()
```

1.4.2 Pivy: Coin3D 's Python wrapping

pivy is Python wrapper of Coin3D C++ lib, via SWIG A new `SoPyScript` Node is added to include Python script directly

1.4.3 Selection of 3D visualization library

OpenCASCADE, as a CAD kernel, did not render 3D object to screen (when FreeCAD was born in 2002) until recently release. Currently, there are several 3D lib based on OpenGL, see a list that works with QT https://wiki.qt.io/Using_3D_engines_with_Qt. 3D gaming engines can also been used to render 3D objects, such as OGRE(Object-Oriented Graphics Rendering Engine), Unreal, Unity.

Selection of Open Inventor to render FreeCAD is based on software license and performance consideration. Open Inventor, originally IRIS Inventor, is a C++ object oriented retained mode 3D graphics API designed by SGI to provide a higher layer of programming for OpenGL. Its main goals are better programmer convenience and efficiency. Open Inventor is free and open-source software, subject to the requirements of the GNU Lesser General Public License (LGPL), version 2.1, in Aug 2000.

Coin3D implements the same API but not source code with Open Inventor, via clean room implementation comapatible Stable release Open Inventor v2.1. Kongsberg ended development of Coin3D in 2011 and released the code under the BSD 3-clause license. It is possible to draw object in OpenInventor Scene by Python, via Coin3D's python wrapper *pivy* , see https://www.freecadweb.org/wiki/Pivy

VTK, is another open source and cross-platform visualisating library, which ParaView is based on. Interoperation is possible, see Method for converting output from the VTK pipeline into Inventor nodes. From 0.17 and beyond, VTK pipeline is added to Fem module.

1.4.4 Discussion of 3D rendering library selection on FreeCAD Forum

Here are my questions on 3D rendering library selection, I posted on FreeCAD Forum:

> I browse OpenCASCADE doc[1], showing graph of OpenCASCADE source code architecture. It is similar with FreeCAD. Why FreeCAD develops its own Foundation Class, Document controller, Object Tree Structure, etc. There are lot of overlapping.
>
> 1) Is that because the license problem? OpenCASCADE is not LGPL compatible during FC startup? Or OpenCASCADE can not support python wrapping function?
>
> 2) OpenCASCADE has visualization for 3D rendering, why OpenInventor/3D is used instead? According to the doc, OCC user interaction is not very strong, but still provide the selection.

[1] http://www.opencascade.com/content/overview

[2] https://forum.freecadweb.org/viewtopic.php?f=10&t=12821&p=102683#p102683 by "ickby"

reply from one key developer:

> First of all FreeCAD works without OpenCASCADE. That is an important feature, not everything needs geometric modeling, for example the Robot Workbench. OCC is only incorporated by the Part Workbench.
>
> Of course one could have based FreeCAD completely on OCC and reuse OCAF and the visalisation, however, there are quite some points against it:
>
> 1. The OCAF overlap is minimal: only some stuff from App could have been reused, the whole GUI handling would have been needed anyway. And to integrate all of the currently available functionality basically the same amount of work would have been needed. According to Jriegel initially FreeCAD based its document structure on OCAF, but it was found to be lacking and then replaced with a custom implementation. And this makes adoptions and enhancements way easier, see for example the recent expression integration.
>
> 2. The OpenCASCADE visualisation was lacking at best over all the years. They put in much work in the last time which significantly improved it, but this was too late for FreeCAD. And the most important issue: OpenCASCADE visualisation is highly tailored towards visualisation of their types. A general interface for arbitrary stuff is not available and hence it is not suited for FreeCAD, where many workbenches draw all kinds of things via the nice openInventor API

1.5 Roadmap of FreeCAD

It is important to track the roadmap of FreeCAD as it is still under heavy development. https://www.freecadweb.org/wiki/Development_roadmap

1.5.1 Keep updated with main components:

The Main external components are upgrade gradually, like OpenInventor, pyCXX.

- C++11 is adopted since 0.17. C++17 latest standard library could replace boost::FileSystem in the future
- Migration from Qt4 to Qt5 is straight-forward (Qt4All.h Switch from Qt4->Qt5) in C++, but depending on availability of LGPL version of Qt5 python wrapping: PySide2
- Python3 support is under implementation
- OpenCASCADE(OCC) and VTK is migrating to 7.0 in late 2016

Transitioning from OpenGL to Vulkan will not happen in the future, while OpenGL should be available for a long time (10 years).

1.5.2 C++11

C++ is NOT a easy language but targeting at high performance, you need to manage memory manually and there are a lot of pitfalls, see http://horstmann.com/cpp/pitfalls.html. Even experienced C++ programmer will find himself/herself has not fully master C++. Exception safety will be beyond layman's brain.

Common pitfalls for c++ are exampled in appendix.

Nevertheless, C++11 is almost a branch new language, or next generation c++. C++11 add some extra keywords like "explicit, overload/final, noexcept" to avoid some unintended mistakes, also introduce new features and extra STL functions like lambda and std::function, constexpr, enum class, smart pointer, auto type derivation, std::thread, atomic, regex etc.

1.5.3 Pyside 2 project for Qt 5.x

According to Qt community news (July, 2016), LGPL python wrapping for Qt 5.x is promising in the near future

> The Pyside 2 project aims to provide a complete port of PySide to Qt 5.x. The development started on GitHub in May 2015. The project managed to port Pyside to Qt 5.3, 5.4 & 5.5. During April 2016 The Qt Company decided to properly support the port (see details).

Chapter 2

Organization of FreeCAD Source Code

The FreeCAD official repository has the standard github project layout, with README, License, Continuous Integration setup files at the repository root. The FreeCAD source tree in the `src/` folder reflects its modular design.

2.1 List of files and folders in FreeCAD source folder

- Base Foundamental classes for FreeCAD

  import as FreeCAD in Python, see detailed description in later section
- App nonGUI code: Document, Property and DocumentObject

  import as FreeCAD in Python, see detailed description in later section
- Gui Qt-based GUI code: macro-recording, Workbench

  import as FreeCADGui in Python, see detailed description in later section
- CXX modified PyCXX containing both python 2 and python 3
- Ext Source code for all modules with each module in one subfolder

  enable module import from FreeCAD to avoid python module name clashing
- Main main() function for FreeCADCmd.exe and FreeCADGui.exe

  "Main() of FreeCADCmd.exe (build up CAD model without GUI but python scripting) and FreeCADGui.exe (Interactive mode)
- Mod Source code for all modules with each module in one subfolder

  Source code of ome modules will be explained in later section
- Tools Tool to build the source code: fcbt.py

  fcbt can generate a basic module from *TEMPLATE* folder,
- Doc Manual and documentation generated by doxygen
- CMakeLists.txt topmost CMake config file, kind of high level cross-platform makefile generator

  Module developer needs not to care about this file, CMakeLists.txt within module will be automatically included.
- FCConfig.h preprocessor shared by all source for portability on diff platforms
- fc.sh export environment variable for CASROOT -> OpenCASCADE

  Module developer needs not to care about this file
- 3rdParty Third party code integration

  boost.CMakeLists.txt CxImage Pivy-0.5 zlib.CMakeLists.txt CMakeLists.txt Pivy salomesmesh
- zipios++ source of zipios++ lib
- Build set the version of FreeCAD
- MacAppBundle config file to generate MacOSX bundle (installer)

- XDGData FreeCAD.desktop file for linux package compliant Linux freedesktop standard
- WindowsInstaller config files to generate windows installer

2.2 List of modules in FreeCAD Mod folder

Mechanical Engineering, CAD and CAM

- Part make primitive 3D objects like cube, cylinder, boolean operation

  The Part module is based on the professional CAD kernel , OpenCasCade, objects and functions.

- OpenSCAD Extra OpenCasCade functions

  use the high level API in Part module instead

- PartDesign modelling complex solid part from 2D sketch

  The Part Design Workbench provides tools for modelling complex solid parts and is based on a Feature editing methodology to produce a single contiguous solid. It is intricately linked with the Sketcher Workbench.

- Draft draw and modify 2D objects, traditional 2D engineering drawing,

  The Draft workbench allows to quickly draw simple 2D objects in the current document, and offers several tools to modify them afterwards. Some of these tools also work on all other FreeCAD objects, not only those created with the Draft workbench. It also provides a complete snapping system, and several utilities to manage objects and settings.

- Drawing put 3D model to paper, can save to DXF and SVG format
- Sketcher build up 3D part from 2D sketch used in PartDesign
- Assembly Assembly of part

  Constraint of

- Cam Computer aided machining (CAM), CNC machining
- Path Tool path for CAM

Civil Engineering

- Idf used by Arch module
- Arch CAD for civil engineering, like design of a house

  The Arch workbench provides modern BIM workflow to FreeCAD, with support for features like IFC support, fully parametric architectural entities such as walls, structural elements or windows, and rich 2D document production. The Arch workbench also feature all the tools from the Draft Workbench

- Ship Build 3D model (hull)for ship

Computer aided engineering (CAE)

- Points points cloud from 3D scanning
- ReverseEngineering build 3D part from points cloud
- Raytracing to render lighting 3D model more vivid as in physical world

  generate photorealistic images of your models by rendering them with an external renderer. The Raytracing workbench works with templates, the same way as the Drawing workbench, by allowing you to create a Raytracing project in which you add views of your objects. The project can then be exported to a ready-to-render file, or be rendered directly.

- MeshPart

- Mesh convert part into triangle mesh for rendering (tessellation)
- Fem Fenite element analysis for part design
- Robot Robot simulator

Utilities

- Plot 2D plot,like XYplot, based on matplotlib

  allows to edit and save output plots created from other modules and tools

- Image import various image format, draw them in 3D plane
- Spreadsheet Excel like data view widget

Testing facility

- Inspection Testing
- Test Workbench for self testing
- Sandbox Testing

Meta workbench

- Web web view of FreeCAD
- Start start page of FreeCAD
- Complete show all toolbar from loadable modules

Module not visible to workbench users

- Import

- JtReader

- Material define standard material property, like density, elastic modulus

  not visible to workbench users, used by Fem module

- TemplatePyMod a collection of python example DocumentObject, ViewProvider

2.3 Build System for FreeCAD

2.3.1 Introduction to CMake

CMake is the cross-platform build tool for FreeCAD. Along with generating the make files that are used to build FC, it also generates the installer for Windows, the DEB/RPM packages for Linux, and an image bundle MacOS X.

See Appendix 2 cmake cheatsheet for a quick start on CMake.

2.3.2 CMake in FreeCAD

The project top level `CMakeLists.txt` at the repository root is very long, detecting third party library detection and dealing compiler and OS differences. In contrast, src/CMakeLists.txt in the source tree is the much shorter, with `add_directory(subfolder_name)`.

CMake folder in the repo root is filled with *.cmake files to detect libraries. If a new workbench with c++ code is added, then CMake's third parties detection cmake file is probably needed.]

```
set(CMAKE_MODULE_PATH "${CMAKE_CURRENT_SOURCE_DIR}/cMake")
```

The following hypothetical code demonstrates how we can specify with CMake to not build OpenFOAM from source but instead we can just install as a binary. The same can be done for other dependencies like other FEM meshing tools like `netgen`, and `gmsh`.

```
if(NOT MSVC)
    OPTION(BUILD_FEM_FOAM "Build the FreeCAD FEM module with the OpenFOAM CFD solver" ON)
else
    OPTION(BUILD_FEM_FOAM "Build the FreeCAD FEM module with the OpenFOAM CFD solver" OFF)
endif(NOT MSVC)
```

Examining further, we can see the following in the the toplevel CMake file src/CMakeLists.txt

```
# -------------------- OpenFOAM --------------------
    if (BUILD_FEM_FOAM)
        find_package(FOAM)
    endif(BUILD_FEM_FOAM)
```

2.3.3 Code Analysis for cMake/SMesh.cMake

```
# Try to find Salome SMESH
# Once complete, this will define
#
# SMESH_FOUND         - if the current system has Salome SMESH
# SMESH_INCLUDE_DIR   - path to the Salome SMESH include directory
# SMESH_LIBRARIES     - path to the Salome SMESH libraries
#


IF (CMAKE_COMPILER_IS_GNUCC)
    FIND_PATH(SMESH_INCLUDE_DIR SMESH_Mesh.hxx
    # These are default search paths, why specify them?
    # /usr/include
    # /usr/local/include
    PATH_SUFFIXES smesh
    )
    FIND_LIBRARY(SMESH_LIBRARY SMESH
    #  /usr/lib
    #  /usr/local/lib
    )
ELSE (CMAKE_COMPILER_IS_GNUCC)
    # Not yet implemented
ENDIF (CMAKE_COMPILER_IS_GNUCC)

SET(SMESH_FOUND FALSE)
IF(SMESH_LIBRARY)
  SET(SMESH_FOUND TRUE)
  GET_FILENAME_COMPONENT(SMESH_LIBRARY_DIR ${SMESH_LIBRARY} PATH)
  set(SMESH_LIBRARIES
    ${SMESH_LIBRARY_DIR}/libDriver.so
    ${SMESH_LIBRARY_DIR}/libDriverDAT.so
    ${SMESH_LIBRARY_DIR}/libDriverSTL.so
    ${SMESH_LIBRARY_DIR}/libDriverUNV.so
    ${SMESH_LIBRARY_DIR}/libSMDS.so
    ${SMESH_LIBRARY_DIR}/libSMESH.so
    ${SMESH_LIBRARY_DIR}/libSMESHDS.so
    ${SMESH_LIBRARY_DIR}/libStdMeshers.so
  )
ENDIF(SMESH_LIBRARY)
```

2.4 Continuous Integration (CI)

Continuous Integration (CI) is basically a paradigm for building and testing automation. CI is crucial to maintain a software project efficiently. There are several CI tools that are utilized by FreeCAD.

2.4.1 Travis CI

Travis is tightly integrated with github and it is widely used in the FreeCAD github repo. For more information on how to set Travis CI for a code repository read this useful guide.

Travis uses YAML files. For the Linux and MacOS platform, the main Travis config file for FreeCAD is .travis.yml.
For Windows, the config file is .travis/build.bat.

2.4.2 Other CI tools

Along with Travis, there is a whole ecosystem of CI Software, you can learn learn more about them through a dedicated Wikipedia page

.circleci/config.yml which will pull a docker image to setup testing environment.

2.4.3 arcconfig and phabricator

```
"phabricator.uri" : "http://pha.geofinder.eu/"
```

2.4.4 vagrant cloud VM

FreeCAD has a subfolder to setup cloud VM for building and testing

2.4.5 docker configuration

subuser/freecad-dev [.subuser.json] adds subuser files for developing freecad within a Docker container.

2.4.6 AppVoyer

Sign up and setup github and bitbucket repo is simple, see official document https://www.appveyor.com/docs/. FreeCAD project use AppVoyer for building on windows, although it is a cross-platform solution.

2.5 Packaging and Deployment

2.5.1 Linux RPM and Deb, AppImage

FreeCAD has been in official repo of major linux distro, but the version may be outdated. For Ubuntu, there is PPA to install the latest or even daily development build. It is recommended for developers to install lattest version, especially Qt5+Python3 combination.

2.5.2 Windows and OSX prebuild

Windows installer is generated by NSIS, and open source windows installer generation tool. Packaging for windows its in another subproject `FreeCADInstProj`, see https://github.com/FreeCAD/FreeCAD/tree/master/src/WindowsInstaller

2.5.3 Other cross platform package managers

The package folder in the repository root contains the `[conda]()` recipe to build Anaconda package. Conda is a popular binary python package (python module compiled from other languages) distribution system.

`fedora` subfolder has the configuration

2.6 Learning path

1. be familiar with FreeCAD Gui Operation as a user

see FreeCAD wiki user hub, tutorials on youtube and user manual

2. be familiar with Python scripting, learning from macro recording.

The amazing feature of FreeCAD is that all GUI operation is recorded in Python console FreeCAD wiki power user hub FreeCAD wiki developer hub

3. be familiar with key classes in FreeCAD source code: Base, App, Gui, Part

It is really challenging to code in C++, Python GIL, Coin3D, OCC. However, it is not needed to know about OCC as module developers. FreeCAD has a online API document for import classes like Properties, Document Objects, see https://www.freecadweb.org/wiki/Power_users_hub#API_Functions

To explore the source code, doxygen generated document is available on `iesensor.com`. Furthermore, `Sourcetrail`can also be one of the tools to help in this effort is a powerful tool, free for non commercial usage, to explore large softwar projects, see FreeCAD forum discussion. The software can be downloaded from https://www.sourcetrail.com/

4. develop/extend pure Python module, the challenging Python wrapping task can be avoided

5. develop/extend hybrid C++ and Python module

6. write 3D rendering code, i.e. ViewProvider derived classes

Chapter 3

Base, App and Main module

In this chapter, the namespace of **Base**, **App** and Main modules are introduced, these 3 modules make a complete program without GUI.
Their functions can be accessed in python by "import FreeCAD", see [FreeCAD module]https://www.freecadweb.org/api/FreeCAD.html

This chapter focused on the property framework and DocumentObject in App namespace, as they are most interesting to module developer. The classes in Base namespace are not frequently used, but understanding of the type system could be useful. Finally, the FreeCAD startup process is tracked in **Main** source code folder.

3.1 List of header files in Base folder

Basic class and Type system

- Type.h register type and create instance from name

  see code snippets in later section

- BaseClass.h using macro function to make type system and link to Python

  see detailed analysis in the later section

- Exception.h base class for all FreeCAD exceptios, derived from BaseClass

  can be constructed from std::exception, see inherit graph for all derived exceptions

Python related

- Interpreter.h Very important and frequently included header file

  define classes: `PyException, PyGILStateLocker, InterpreterSingleton` define methods: `addType()`, `loadModule()`, will be discussed in Python wrapping section

- PyExport.h define PyHandle<> temaplate class

  Using pointers on classes derived from PyObjectBase would be potentionaly dangerous because you would have to take care of the reference counting of python by yourself. Therefore this class was designed. It takes care of references and as long as a object of this class exists the handled class get not destructed. That means a PyObjectBase derived object you can only destruct by destructing all FCPyHandle and all python references on it!

- PyObjectBase.h Base Class for all classed exposed to python interpreter
- PyTools.h ppembed-modules.c: load,access module objects
- swigrun.cpp

  cpp files related to diff swig version are not listed here

- swigrun.inl swig for python binding

Input and output and File related

- Reader.h XML file reader for DocumentObject for persistence
- Writer.h XML file writer for DocumentObject

- Stream.h define adapter classes for Qt class QByteArray; class QIODevice;class QBuffer;
- InputSource.h

  `class BaseExport StdInputStream : public XERCES_CPP_NAMESPACE_QUALIFIER BinInputStream`

- FileInfo.h File name unification class

  This class handles everything related to file names the file names which are internal generally UTF-8 encoded on all platforms.

- FileTemplate.h used for testing purpose
- gzStream.h gzip compressed file Stream
- Console.h output message to terminal which starts FreeCADCmd

  ConsoleObserver and ConsoleSingleton with python code [Console.cpp], This is not Python Console, but dealing with stdio, logging to terminal which starts FreeCADCmd. `class BaseExport ConsoleObserverStd: public ConsoleObserver` to write Console messages and logs the system con.

- Parameter.h ParameterGrp: key-value, XML persistence as app config

  ```
  class  BaseExport ParameterGrp    : public Base::Handled,public Base::Subject <const char*>
  class BaseExport ParameterManager  : public ParameterGrp
  ```

- Debugger.h Debugger class

  Debugging related classes in source files [Debugger.h, Debugger.cpp, StackWalker.h, StackWalker.cpp, MemDebug.h]

serialization support, example of class with cpp, py and XML code

- Persistence.h serialization of objects

  base class for DocumentObject, Property, etc

- Persistence.cpp C++ implementation of Persistence class
- PersistencePyImp.cpp automatically generated C++ code for exporting Persistence class to python
- PersistencePy.xml XML to generate PersistencePyImp.cpp by python script

Geometry related calculation classes with *Py.cpp

- Axis.h Class: Axis
- BoundBox.h bounding boxes of the 3D part, define max{x,y,z} and min{x,y,z}
- Rotation.h define class and method for rotation an objecti n 3D space
- Placement.h class to place/relocate an object in 3D space

  see official api doc: https://www.freecadweb.org/api/Placement.html

- Vector.hTemplate class represents a point, direction in 3D space `typedef Vector3<float>  Vector3f; typedef Vector3<double> Vector3d;`
- Matrix.hTemplate class: Matrix4D for coordination translation and rotation

- GeometryPyCXX.h template class GeometryT<>

  This is a template class to provide wrapper classes for geometric classes like `Base::Matrix4D, Base::Rotation Placement and Base::BoundBox` . Since the class inherits from `Py::Object` it can be used in the same fashion as `Py::String, Py::List`, etc. to simplify the usage with them.

Geometry related classes without *Py.cpp

- CordiniateSystem.h XYZ only?

  local cylindral coordination is common

- ViewProj.h View Projection
- Builder3D.h class Builder3D, InventorBuilder

  A Builder class for 3D representations without the visual representation of data. Nevertheless it's often needed to see some 3D information, e.g. points, directions, when you program or debug an algorithm. For that purpose Builder3D was made. This class allows you to build up easily a 3D representation of some math and lgorithm internals. You can save this representation to a file and see it in an Inventor viewer, or put it to the log.

- Tools2D.h class Vector2D, BoundBox2D,Polygon2D, Line2D

Unit and physical quantity

- Unit.h Physical unit like Newton, second for time

  `struct  UnitSignature{9* int32_t}` International System of Units (SI) has only 7 base unit boost has its unit system; OpenFoam also has its templated class for physical quantity. OpenFOAM uses a unit tuple of 7 foundamental SI base unit

- UnitScheme.h Base class for diff schemes like imperial, SI MKS(meter, kg, second) ,etc

  [Unit.cpp, UnitsApi.cpp, UnitsSchema.h, UnitsSchemaInternal.h, Unit.h UnitsApi.h, UnitsSchemaImperial1.cpp, UnitsSchemaMKS.cpp UnitPyImp.cpp, UnitsApiPy.cpp, UnitsSchemaImperial1.h, UnitsSchemaMKS.h UnitPy.xml, UnitsSchema.cpp, UnitsSchemaInternal.cpp]

- Quantity.h define static quantity with unit like Force

Important utility classes

- TimeInfo.h helper class to deal with time_t, currentDataTimeString()
- Base64.h text encoding helper class for URL
- Uuid.h a wrapper of QUuid class: unique ID 128bit
- Handle.h class Base::Handled, Base::Reference: Reference counting pattern

  Implementation of the reference counting pattern. Only able to instantiate with a class inheriting Base::Handled.

- Factory.h Factory design pattern to create object

  to get the singleton instance of concrete class: `ScriptFactorySingleton & ScriptFactorySingleton::Instance (   void         )`

- Observer.h Observer design pattern: define class Subject,Observer

  `template <class MessageType> class Subject;`

- Sequencer.h report Progress

  ConsoleSequencer, EmptySequencer

- FutureWatcherProgress.h progress report based on sequencer

  it is derived from QObject, so can be used in Qt object event loop

- Tools.h Main dealing with string encoding, std::string <-> QString
- XMLtools.h include Xerces library header

Note: some class can be found in C++11/17 standard library or boost

+Swap : std::swap + Handle : shared pointer + FileInfo: Filesystem C++17 + Unit and quality: boost + TimeInfo: QDataTime, C++11 to get 64bit timestamp

3.1.1 Frequently included headers files

```
#include <Base/Console.h>            // PrintMessage(), singleton for logging
#include <Base/Tools.h>                // string encoding conversion utf <-> QString
#include <Base/Interpreter.h>        // python interpreter
#include <Base/Exception.h>          // all exception should be derived from Base::Exception
```

3.1.2 Correct way of using *Sequencer* in try-catch block

```
#include <Base/Sequencer.h>
void runOperation();
void myTest()
{
  try{
     runOperation();
  } catch(...) {
      // the programmer forgot to halt the sequencer here
      // If SequencerLauncher leaves its scope the object gets destructed automatically and
      // stops the running sequencer.
  }
}
void runOperation()
{
  // create an instance on the stack (not on any terms on the heap)
  SequencerLauncher seq("my text", 10);
  for (int i=0; i<10; i++)
  {
    // do something (e.g. here can be thrown an exception)
    ...
    seq.next ();
  }
}
```

3.1.3 String encoding utf8 and conversion into wchar_t QString

The string encoding for FreeCAD is different form Qt's wide char, using the helper functions in src/Base/Tools.h

`fromStdString(const std::string & s)` and `toStdString(const QString& s)`

```
struct BaseExport Tools
{
    static std::string getUniqueName(const std::string&, const std::vector<std::string>&,int d=0);
    static std::string addNumber(const std::string&, unsigned int, int d=0);
    static std::string getIdentifier(const std::string&);
    static std::wstring widen(const std::string& str);
    static std::string narrow(const std::wstring& str);
    static std::string escapedUnicodeFromUtf8(const char *s);
    /**
     * @brief toStdString Convert a QString into a UTF-8 encoded std::string.
     * @param s String to convert.
     * @return A std::string encoded as UTF-8.
     */
    static inline std::string toStdString(const QString& s) { QByteArray tmp = s.toUtf8(); return std::string(

    /**
     * @brief fromStdString Convert a std::string encoded as UTF-8 into a QString.
     * @param s std::string, expected to be UTF-8 encoded.
     * @return String represented as a QString.
     */
    static inline QString fromStdString(const std::string & s) { return QString::fromUtf8(s.c_str(), s.size())

}
```

3.2 Type, BaseClass, PyObjectBase

It is important for c++ framework is have a root base class, thereby, essential functions like reference counting, runtime type information is implemented. *QObject* for Qt is the best example.

BaseClass and Type can be reused in other projects (my example will be added soon, email me if it is not available)

3.2.1 Type system

Just like `Base:Unit` class, type info is saved to a struct *TypeData* see src/Base/Type.h

```
struct Base::TypeData
{
  TypeData(const char *theName,
           const Type type = Type::badType(),
           const Type theParent = Type::badType(),
           Type::instantiationMethod method = 0
          ):name(theName),parent(theParent),type(type),instMethod(method) { }

  std::string name;
  Type parent;
  Type type;
  Type::instantiationMethod instMethod;
};

class Type
{
  //...
  static void *createInstanceByName(const char* TypeName, bool bLoadModule=false);

  static int getAllDerivedFrom(const Type type, std::vector<Type>& List);

  static int getNumTypes(void);

  static const Type createType(const Type parent, const char *name,instantiationMethod method = 0);
private:
  unsigned int index;
  static std::map<std::string,unsigned int> typemap;
  static std::vector<TypeData*>     typedata;

  static std::set<std::string>  loadModuleSet;
}
```

3.2.2 src/Base/BaseClass.h

Macro function is widely employed to generate boilplate code, similar with QObject macro for QT

```
#ifndef BASE_BASECLASS_H
#define BASE_BASECLASS_H

#include "Type.h"

// Python stuff
typedef struct _object PyObject;


/// define for subclassing Base::BaseClass
#define TYPESYSTEM_HEADER() \
public: \
  static Base::Type getClassTypeId(void); \
  virtual Base::Type getTypeId(void) const; \
  static void init(void);\
  static void *create(void);\
private: \
  static Base::Type classTypeId


/// define to implement a  subclass of Base::BaseClass
#define TYPESYSTEM_SOURCE_P(_class_) \
Base::Type _class_::getClassTypeId(void) { return _class_::classTypeId; } \
Base::Type _class_::getTypeId(void) const { return _class_::classTypeId; } \
Base::Type _class_::classTypeId = Base::Type::badType();  \
void * _class_::create(void){\
   return new _class_ ();\
}

/// define to implement a  subclass of Base::BaseClass
#define TYPESYSTEM_SOURCE_ABSTRACT_P(_class_) \
Base::Type _class_::getClassTypeId(void) { return _class_::classTypeId; } \
Base::Type _class_::getTypeId(void) const { return _class_::classTypeId; } \
Base::Type _class_::classTypeId = Base::Type::badType();  \
void * _class_::create(void){return 0;}


/// define to implement a subclass of Base::BaseClass
#define TYPESYSTEM_SOURCE(_class_, _parentclass_) \
TYPESYSTEM_SOURCE_P(_class_);\
void _class_::init(void){\
  initSubclass(_class_::classTypeId, #_class_ , #_parentclass_, &(_class_::create) ); \
}

/// define to implement a subclass of Base::BaseClass
#define TYPESYSTEM_SOURCE_ABSTRACT(_class_, _parentclass_) \
TYPESYSTEM_SOURCE_ABSTRACT_P(_class_);\
void _class_::init(void){\
  initSubclass(_class_::classTypeId, #_class_ , #_parentclass_, &(_class_::create) ); \
}

namespace Base
{
/// BaseClass class and root of the type system
class BaseExport BaseClass
{
public:
  static Type getClassTypeId(void);
  virtual Type getTypeId(void) const;
  bool isDerivedFrom(const Type type) const {return getTypeId().isDerivedFrom(type);}

  static void init(void);

  virtual PyObject *getPyObject(void);
  virtual void setPyObject(PyObject *);

  static void *create(void){return 0;}
private:
  static Type classTypeId;
protected:
  static void initSubclass(Base::Type &toInit,const char* ClassName, const char *ParentName, Type::instantiati

public:
  /// Construction
  BaseClass();
  /// Destruction
  virtual ~BaseClass();

};
} //namespace Base

#endif // BASE_BASECLASS_H

/////////////////////////////////////////////////////////////////////////////
#include "PreCompiled.h"

#ifndef _PreComp_
# include <assert.h>
#endif

/// Here the FreeCAD includes sorted by Base,App,Gui......
#include "BaseClass.h"
#include "PyObjectBase.h"

using namespace Base;

Type BaseClass::classTypeId = Base::Type::badType();


//**************************************************************************
// separator for other implementation aspects

void BaseClass::init(void)
{
    assert(BaseClass::classTypeId == Type::badType() && "don't init() twice!");
    /* Make sure superclass gets initialized before subclass. */
    /*assert(strcmp(#_parentclass_), "inherited"));*/
    /*Type parentType(Type::fromName(#_parentclass_));*/
    /*assert(parentType != Type::badType() && "you forgot init() on parentclass!");*/

    /* Set up entry in the type system. */
    BaseClass::classTypeId =
        Type::createType(Type::badType(),
                         "Base::BaseClass",
                         BaseClass::create);
}

Type BaseClass::getClassTypeId(void)
{
    return BaseClass::classTypeId;
}

Type BaseClass::getTypeId(void) const
{
    return BaseClass::classTypeId;
}


void BaseClass::initSubclass(Base::Type &toInit,const char* ClassName, const char *ParentName,
                             Type::instantiationMethod method)
{
    // don't init twice!
    assert(toInit == Base::Type::badType());
    // get the parent class
    Base::Type parentType(Base::Type::fromName(ParentName));
    // forgot init parent!
    assert(parentType != Base::Type::badType() );

    // create the new type
    toInit = Base::Type::createType(parentType, ClassName, method);
}

/**
 * This method returns the Python wrapper for a C++ object. It's in the responsibility of
 * the programmer to do the correct reference counting. Basically there are two ways how
 * to implement that: Either always return a new Python object then reference counting is
 * not a matter or return always the same Python object then the reference counter must be
 * incremented by one. However, it's absolutely forbidden to return always the same Python
 * object without incrementing the reference counter.
 *
 * The default implementation returns 'None'.
 */
PyObject *BaseClass::getPyObject(void)
{
    assert(0);
    Py_Return;
}

void BaseClass::setPyObject(PyObject *)
{
    assert(0);
}
```

3.2.3 src/Base/PyObjectBase.h

Py_Header is a macro function, `PyObject` is defined in `<python.h>`, the header for python C API.

```
/** The PyObjectBase class, exports the class as a python type
 *  PyObjectBase is the base class for all C++ classes which
 *  need to get exported into the python namespace.

class BaseExport PyObjectBase : public PyObject
{
    /** Py_Header struct from python.h.
     *  Every PyObjectBase object is also a python object. So you can use
     *  every Python C-Library function also on a PyObjectBase object
     */
    Py_Header

// The definition of Py_Header:
//This must be the first line of each PyC++ class
#define Py_Header                                            \
public:                                                      \
    static PyTypeObject   Type;                              \
    static PyMethodDef    Methods[];                         \
    static PyParentObject Parents[];                         \
    virtual PyTypeObject *GetType(void) {return &Type;}      \
    virtual PyParentObject *GetParents(void) {return Parents;}
```

3.2.4 src/Base/Persistence.h

save and restore into XML string

3.2.5 GeoFeature: Base class of all geometric document objects

```
void GeoFeature::transformPlacement(const Base::Placement &transform)
{
    Base::Placement plm = this->Placement.getValue();
    plm = transform * plm;
    this->Placement.setValue(plm);
}
```

3.3 Unit scheme for physical quantity

Define 3 unit schemes: Internal, SI (MKS) and imperial unit system and conversion

3.3.1 src/Base/Unit.h

There are 7 SI base units, but FreeCAD defined *Density*, which is a derived unit

```
struct UnitSignature{
    int32_t Length:UnitSignatureLengthBits;
    int32_t Mass:UnitSignatureMassBits;
    int32_t Time:UnitSignatureTimeBits;
    int32_t ElectricCurrent:UnitSignatureElectricCurrentBits;
    int32_t ThermodynamicTemperature:UnitSignatureThermodynamicTemperatureBits;
    int32_t AmountOfSubstance:UnitSignatureAmountOfSubstanceBits;
    int32_t LuminoseIntensity:UnitSignatureLuminoseIntensityBits;
    int32_t Angle:UnitSignatureAngleBits;
    int32_t Density:UnitSignatureDensityBits;
};
```

Predefined static Unit types: `static Unit Length; ... static Unit Stress;`

3.3.2 src/Base/Quantity.h

Quantity is value + unit. Common quantities defined as static instances. Quantity string can be parsed into value and unit by *quantitylexer*

3.4 List of header files in App folder

Application

- Application.h method called in QApplication to init and run FreeCAD

  Mange `App::Document`, import/export files ,Path, ParameterManager/config, init()/addTypes() The FreeCAD startup process will call `App::Application::initApplication()`, setup/init FreeCAD python module

- ApplicationPy.cpp export method to python as FreeCAD module

  `import FreeCAD dir(FreeCAD)`

- Branding.h.cpp Customise splashscreen and banner in CMD mode
- FreeCADInit.py def InitApplications() which adds Mod path to sys.path

  prepend all module paths to Python search path Searching for modules... by file: `Init.py` in each module folder `FreeCAD.__path__ = ModDict.values()` init every application by `import Init.py`, call `InitApplications()`

Property framework

- Property.h Base class for all Properties, derived from Base::Persistence

  Can access attributes of a class by name without knowing the class type, enable access in Python, parameterise 3D part,
  Useful methods: get/setValue(), save/restore(), get/setPyObject(), copy/paste(), getGroup/getPath/getType/getDocumentation(

[PropertyContainer.cpp, PropertyFile.cpp, PropertyPythonObject.cpp PropertyContainer.h, PropertyFile.h, PropertyPythonObject.h PropertyContainerPyImp.cpp, PropertyGeo.cpp, PropertyStandard.cpp PropertyContainerPy.xml, PropertyGeo.h, PropertyStandard.h Property.cpp, Property.h, PropertyUnits.cpp PropertyExpressionEngine.cpp, PropertyLinks.cpp, PropertyUnits.h PropertyExpressionEngine.h, PropertyLinks.h]

- PropertyStandard.h define Property for common types like string int

  why not template in c++?

- PropertyContainer.h define class PropertyContainer and PROPERTY related macro functions

  DocumentObject is derived from this class, macro function will be explained in Property framework section

- DynamicProperty.h Runtime added into PropertyContainer

  not derived from App::Property

- ObjectIdentifier.h define Component class and ObjectIdentifier class

  A component is a part of a Path object, and is used to either name a property or a field within a property. A component can be either a single entry, and array, or a map to other sub-fields.

- PropertyLinks.h property is to Link DocumentObjects and Features in a document.
- PropertyUnits.h Quantiy as Property, PropertyAngle, PropertyAcceleration, etc

  its path is based on ObjectIdentifier

- PropertyPythonObject.h to manage Py::Object instances as properties
- PropertyGeo.h PropertyVector, PropertyMatrix, Property

  PropertyPlacementLink, `class AppExport PropertyGeometry : public App::Property // transformGeometry() getBoundBox3d() class AppExport PropertyComplexGeoData : public App::PropertyGeometry`

- Enumeration.h A bidirectional stringinteger mapping for enum

App::Document and App::DocumentObject

- Document.h Corresponding to FreeCAD main saving file format for 3D part or other info: *.FCstd

  [Document.cpp, DocumentObject.h Document.h, DocumentObjectPyImp.cpp DocumentObject.cpp, DocumentObjectPy.xml DocumentObjectFileIncluded.cpp, DocumentObserver.cpp DocumentObjectFileIncluded.h, DocumentObserver.h DocumentObjectGroup.cpp, DocumentObserverPython.cpp DocumentObjectGroup.h, DocumentObserverPython.h DocumentObjectGroupPyImp.cpp, DocumentPyImp.cpp DocumentObjectGroupPy.xml, DocumentPy.xml]

- DocumentObject.h Most important class in FreeCAD

  The inheritance chain is: Base::BaseClass->Base::Persistence->Base::PropertyContainer->DocumentObject

- DocumentGroup.h DocumentObjectGroup class: Container of DocumentObject
- DocumentObserver.h Minitoring the create, drop, change of DocumetnObject and emit signal
- MergeDocuments.h helper classes for document merge
- Transactions.h A collection of operation on DocumentObject like SQL database that can be rolled back

  DocumentObject could be restored to a previous state

- FeaturePython.h Generic Python feature class which allows to behave every DocumentObject derived class as Python feature simply by subclassing

  `// Special Feature-Python classes,  Feature is another name for DocumentObject typedef FeaturePythonT<Docu`
  `FeaturePython; typedef FeaturePythonT<GeoFeature> GeometryPython;`

Expression framework

- Expression.h Base class for FunctionExpression, OperatorExpression etc.

  expression and Parser for parameterization [Expression.cpp, ExpressionParser.tab.c, lex.ExpressionParser.c Expression.h, ExpressionParser.tab.h, PropertyExpressionEngine.cpp ExpressionParser.l, ExpressionParser.y, PropertyExpressionEngine.h]

Extension framework

- Extension.h extend function of object other than inheritance

[DocumentObjectExtension.cpp, GeoFeatureGroupExtension.cpp, DocumentObjectExtension.h, GeoFeatureGroupExtension.h DocumentObjectExtensionPyImp.cpp, GeoFeatureGroupExtensionPyImp.cpp, DocumentObjectExtensionPy.xml, GeoFeatureGroupExtensionPy.xml ExtensionContainer.cpp, GroupExtension.cpp, ExtensionContainer.h, GroupExtension.h ExtensionContainerPyImp.cpp, GroupExtensionPyImp.cpp, ExtensionContainerPy.xml, GroupExtensionPy.xml Extension.cpp, OriginGroupExtension.cpp, Extension.h, OriginGroupExtension.h ExtensionPyImp.cpp, OriginGroupExtensionPyImp.cpp, ExtensionPy.xml, OriginGroupExtensionPy.xml]

Utilities

- MeasureDistance.h Measure distance between two entity
- ColorModel.h Color bar like grayscale, inverse gray scale, Tria,

  Color class is defined here, constructed from uint32_t or 4 float number for RGBA.

- Material.h appearance: color and transparency for rendering of 3D object

  define a few standard material MaterialObject is derived from DocumentObject and contains data from Material class. [Material.cpp, MaterialObject.cpp, MaterialPyImp.cpp, Material.h, MaterialObject.h, MaterialPy.xml]

- MaterialObject.h DocumentObject store key-valve pair for material information

  physical property of *.ini style FCMat files, under `src/Mod/Material/StandardMaterial/<MaterialName>.FCMat` `Fem::MechanicalMaterial` is python class derived from this class

App::GeoFeature and derived classes

- GeoFeature.h Base class of all geometric document objects

  Derived from `DocumentObject`, contains only *PropertyPlacement*, see [GeoFeature.cpp]

- Plane.h Object Used to define planar support for all kind of operations in the document space

  sketch is done on planes, derived from `App::GeoFeature` which is derived from `DocumentObject`

- Placement.h define six degree of freedom (orientation and position) for placing a part in space

  derived from `App::GeoFeature`, A placement defines an orientation (rotation) and a position (base) in 3D space. It is used when no scaling or other distortion is needed.

- InventorObject.h derived from App::GeoFeature with only 2 properties: PropertyString Buffer, FileName;
- VRMLObject.h derived from App::GeoFeature

App::Data namespace and ComplexGeoData class

- ComplexGeoData.h store data to represent complex geometry in line, facet(triangle) and segment

  declare `Segment`, and `ComplexGeoData`, which has ref counting, in `App::Data` namespace. `class AppExport ComplexGeoData: public Base::Persistence, public Base::Handled`

3.5 Property framewrok

see Doxygen generated document for example of using the property framework. However, module developer needs not to know such low level details. It's like the reflection mechanism of Java or C#. This ability is introduced by the App::PropertyContainer class and can be used by all derived classes.

This makes it possible in the first place to make an automatic mapping to python (e.g. in App::FeaturePy) and abstract editing properties in Gui::PropertyEditor.

Access property in Python is easier than c++, which needs a `static_cast<>` for type conversion (downcast)

`App::Property* PropertyContainer::getPropertyByName(const char *name) const` works for property defined in c++ and python

Adding property could be done via ADD_PROPERTY macro or adding dynamic property

```
virtual App::Property *     addDynamicProperty (const char *type, const char *name=0,
     const char *group=0, const char *doc=0, short attr=0, bool ro=false, bool hidden=false)
```

3.5.1 Naming of property and PropertyEditor

Property name should begin with uppercase like "ThisPropertyName", and it will show as "This Property Name" in property editor There is indeed the logic to split property names on capital letters and insert a space. But that's only for visual purposes and doesn't affect changing a property value.

3.5.2 src/App/PropertyStandard.h

Define property for common C++ data type: PropertyBool, PropertyInteger (long), PropertyString (utf8/std::string), PropertyFloat (double), PropertyPath (boost::filesystem::path), PropertyFont, PropertyColor, PropertyMaterial,PropertyUuid, PropertyStringLists, PropertyMap(std::map)

PropertyIntegerConstraint is PropertyInteger with upper and lower Bound.

```
struct Constraints { long LowerBound, UpperBound, StepSize; };
void setConstraints(const Constraints* sConstraint); /// get the constraint struct const Constraints* getConst
```

For *PropertyList* derived class, it is possible to set and get values in `std::vector<>` reference `setValues(std::vector<T>&)`, std::vector& getValues()‘

PropertyFloatList actually holds double precision data (float64)

3.5.3 *PropertyEnumeration*, see src/App/Enumeration.h

App::Enumeration as the private data structure to hold this enumeration property

- setEnums() Accept NULL ended string array
- `const char * getValueAsString(void) const;`

It can be used with Combobox in PropertyEditor

see example in src/Mod/Fem/App/FemMeshShapeNetgenObject.cpp

```
#include <App/PropertyStandard.h>
const char* FinenessEnums[]= {"VeryCoarse","Coarse","Moderate","Fine","VeryFine","UserDefined",NULL};
...
ADD_PROPERTY_TYPE(Fineness,(2), "MeshParams",Prop_None,"Fineness level of the mesh");
Fineness.setEnums(FinenessEnums);
```

3.5.4 Geometry related property

PropertyVector, PropertyMatrix, PropertyPlacement: which can be accessed in property editor

see src/App/PropertyGeo.cpp

`PropertyPlacementLink : public PropertyLink` `PropertyComplexGeoData : public App::PropertyGeometry`

3.5.5 File related property

see classes defined in src/App/PropertyFile.cpp

```
App::PropertyPath
App::PropertyFile
App::PropertyFileIncluded
App::PropertyPythonObject
```

3.5.6 Links related property

src/App/PropertyLinks.cpp link to other document object in the this document. For example, FemMeshObject has a link to Part object.

There are scennarios where link to sub feature are needed, e.g. faces of a part.

3.5.7 PropertyMap

implements a key/value list as property. The key ought to be ASCII the Value should be treated as UTF8 to be save

3.5.8 Properties with Units for physical Quantities

src/App/PropertyUnits.cpp

```
TYPESYSTEM_SOURCE(App::PropertyDistance, App::PropertyQuantity);

PropertyDistance::PropertyDistance()
{
     setUnit(Base::Unit::Length);
}
```

3.5.9 src/App/Property.h

```
/// Set value of property
    virtual void setPathValue(const App::ObjectIdentifier & path, const boost::any & value);
    /// Get value of property
    virtual const boost::any getPathValue(const App::ObjectIdentifier & path) const;
...
    /** Status bits of the property
     * The first 8 bits are used for the base system the rest can be used in
     * descendent classes to to mark special statuses on the objects.
     * The bits and their meaning are listed below:
     * 0 - object is marked as 'touched'
     * 1 - object is marked as 'immutable'
     * 2 - object is marked as 'read-ony' (for property editor)
     * 3 - object is marked as 'hidden' (for property editor)
     */
    std::bitset<32> StatusBits;
...
private:
    PropertyContainer *father;
```

note: `boost::any` and `boost::filesystem::path` will be included into C++17.

3.5.10 src/App/PropertyContainer.h

```
enum PropertyType
{
  Prop_None     = 0,
  Prop_ReadOnly = 1,
  Prop_Transient= 2,
  Prop_Hidden   = 4,
  Prop_Output   = 8
};

struct AppExport PropertyData
{
  struct PropertySpec
  {
    const char* Name;
    const char * Group;
    const char * Docu;
    short Offset,Type;
  };
  // vector of all properties
  std::vector<PropertySpec> propertyData;
  const PropertyData *parentPropertyData;

  void addProperty(const PropertyContainer *container,const char* PropName, Property *Prop, const char* Proper

  const PropertySpec *findProperty(const PropertyContainer *container,const char* PropName) const;
  const PropertySpec *findProperty(const PropertyContainer *container,const Property* prop) const;

  const char* getName         (const PropertyContainer *container,const Property* prop) const;
  short       getType         (const PropertyContainer *container,const Property* prop) const;
  short       getType         (const PropertyContainer *container,const char* name)     const;
  const char* getGroup        (const PropertyContainer *container,const char* name)     const;
  const char* getGroup        (const PropertyContainer *container,const Property* prop) const;
  const char* getDocumentation(const PropertyContainer *container,const char* name)     const;
  const char* getDocumentation(const PropertyContainer *container,const Property* prop) const;

  Property *getPropertyByName(const PropertyContainer *container,const char* name) const;
  void getPropertyMap(const PropertyContainer *container,std::map<std::string,Property*> &Map) const;
  void getPropertyList(const PropertyContainer *container,std::vector<Property*> &List) const;
};

class AppExport PropertyContainer: public Base::Persistence
{
private:
  // forbidden
  PropertyContainer(const PropertyContainer&);
  PropertyContainer& operator = (const PropertyContainer&);

private:
  static PropertyData propertyData;
};
```

3.5.11 Macro functions for Property

- PROPERTY_HEADER has included the TYPESYSTEM_HEADER(), so it is added to type system automatically.
- ADD_PROPERTY(*prop*, *defaultval*) used in cpp file
- ADD_PROPERTY_TYPE(*prop*, *defaultval*, *group*,*type*,*Docu*), where *Docu* is docstring tooltip for user, *group* should be "Data" , *type* is enum PropertyType, Prop_None is the most common type
- PROPERTY_SOURCE(*class*, *parentclass*) used in cpp file, first line of constructor
- PROPERTY_SOURCE_ABSTRACT,
- TYPESYSTEM_SOURCE_TEMPLATE(*class*),
- PROPERTY_SOURCE_TEMPLATE(*class*, *parentclass*

3.6 Document-View-Observer Pattern

App::Document, **Gui::ViewProvider**, **App::DocumentObserver**

3.6.1 src/App/Document.h

- `class AppExport Document : public App::PropertyContainer`

  contains CAD model's meta info as property: Author, Date, license, etc.
- contains *DocumentObjectGroup* which is container of DocumentObject
- save and load to native FreeCAD file format: zipped folder of `Property<T>` XML nodes, PropertyLink (path)
- File export and import function, register all the supported importable file types
- `addDocumentObject()/remDocumentObject()`
- Transaction support as in database: Undo
- recompute():
- viewProvider: update view in 3D scene

3.6.2 src/App/DocumentObject.h

`class AppExport DocumentObject: public App::PropertyContainer` , Base class of all Classes handled in the Document.

see https://www.freecadweb.org/api/DocumentObject.html, some important methods (excluding methods from `App::PropertyContainer`) are extracted here:

- `state` enumeration.

```
enum    ObjectStatus {
  Touch = 0, Error = 1, New = 2, Recompute = 3,
  Restore = 4, Expand = 16
}
```

- `__setstate__(value)` allows to save custom attributes of this object as strings, so they can be saved when saving the FreeCAD document
- `touch()` marks this object to be recomputed
- `purgeTouched()` removes the to-be-recomputed flag of this object
- `execute()` this method is executed on object creation and whenever the document is recomputed

Implementation: [src/App/DocumentObject.h] and [src/App/DocumentObject.cpp]

```
protected:
    /* get called by the document to recompute this feature
      * Normally this method get called in the processing of Document::recompute().
      * In execute() the output properties get recomputed with the data from linked objects and objects own  p
      */
    virtual App::DocumentObjectExecReturn *execute(void);

    /* Status bits of the document object
     * The first 8 bits are used for the base system the rest can be used in
     * descendent classes to to mark special statuses on the objects.
     * The bits and their meaning are listed below:
     *  0 - object is marked as 'touched'
     *  1 - object is marked as 'erroneous'
     *  2 - object is marked as 'new'
     *  3 - object is marked as 'recompute', i.e. the object gets recomputed now
     *  4 - object is marked as 'restoring', i.e. the object gets loaded at the moment
     *  5 - reserved
     *  6 - reserved
     *  7 - reserved
     * 16 - object is marked as 'expanded' in the tree view
     */
    std::bitset<32> StatusBits;
```

```
protected: // attributes
    Py::Object PythonObject;
    /// pointer to the document this object belongs to
    App::Document* _pDoc;
    // Connections to track relabeling of document and document objects
    boost::BOOST_SIGNALS_NAMESPACE::scoped_connection onRelabledDocumentConnection;
    boost::BOOST_SIGNALS_NAMESPACE::scoped_connection onRelabledObjectConnection;

    /// Old label; used for renaming expressions
    std::string oldLabel;

    // pointer to the document name string (for performance)
    const std::string *pcNameInDocument;
```

3.6.3 Observer and Subject pattern for documentObject

DocumentObserver class and DocumentObjectObserver class monitor change/add/remove of Document/DocumentObject and trigger slotFunction()

```
template <class MessageType> class Subject;
template <class _MessageType> class Observer
```

Observer class Implementation of the well known Observer Design Pattern. * The observed object, which inherit FCSubject, will call all its observers in case of changes. A observer class has to attach itself to the observed object.

The DocumentObserver class simplfies the step to write classes that listen to what happens inside a document. This is very useful for classes that needs to be notified when an observed object has changed.

```
    void attachDocument(Document*);
    /* Checks if the given document is about to be opened/closed */
    virtual void slotDeletedDocument(const App::Document& Doc)        {}
    /* Checks if a new object was added, removed, changed. */
    virtual void slotCreatedObject(const App::DocumentObject& Obj) {}
```

3.6.4 App::DocumentObjectExecReturn

defined in file src/App/DocumentObject.h

```
/** Return object for feature execution
*/
class AppExport DocumentObjectExecReturn
{
public:
    DocumentObjectExecReturn(const std::string& sWhy, DocumentObject* WhichObject=0)
        : Why(sWhy), Which(WhichObject)
    {
    }
    DocumentObjectExecReturn(const char* sWhy, DocumentObject* WhichObject=0)
        : Which(WhichObject)
    {
        if(sWhy)
            Why = sWhy;
    }

    std::string Why;
    DocumentObject* Which;
};
```

3.6.5 FeaturePython

```
DocumentObjectExecReturn *FeaturePythonImp::execute()
{
    // Run the execute method of the proxy object.
    Base::PyGILStateLocker lock;
    try {
        Property* proxy = object->getPropertyByName("Proxy");
        if (proxy && proxy->getTypeId() == PropertyPythonObject::getClassTypeId()) {
            Py::Object feature = static_cast<PropertyPythonObject*>(proxy)->getValue();
            if (feature.hasAttr("__object__")) {
                Py::Callable method(feature.getAttr(std::string("execute")));
                Py::Tuple args;
                method.apply(args);
            }
            else {
                Py::Callable method(feature.getAttr(std::string("execute")));
                Py::Tuple args(1);
                args.setItem(0, Py::Object(object->getPyObject(), true));
                method.apply(args);
            }
        }
    }
    catch (Py::Exception&) {
        Base::PyException e; // extract the Python error text
        e.ReportException();
        std::stringstream str;
        str << object->Label.getValue() << ": " << e.what();
        return new App::DocumentObjectExecReturn(str.str());
    }

    return DocumentObject::StdReturn;
}
```

3.6.6 FeaturePythonPy template class

This template helps to expose document object derived class to python as/like DocumentObjectPy see chapters on Fem module code analysis and python wrapping for details

```
template <class FeaturePyT>
class FeaturePythonPyT : public FeaturePyT
{
public:
    static PyTypeObject   Type;
    static PyMethodDef    Methods[];

public:
    FeaturePythonPyT(DocumentObject *pcObject, PyTypeObject *T = &Type);
    virtual ~FeaturePythonPyT();

    /** @name callbacks and implementers for the python object methods */
    //@{
    static  int __setattr(PyObject *PyObj, char *attr, PyObject *value);
    /// callback for the addProperty() method
    static PyObject * staticCallback_addProperty (PyObject *self, PyObject *args);
    /// implementer for the addProperty() method
    PyObject*  addProperty(PyObject *args);
    /// callback for the removeProperty() method
    static PyObject * staticCallback_removeProperty (PyObject *self, PyObject *args);
    /// implementer for the removeProperty() method
    PyObject*  removeProperty(PyObject *args);
    /// callback for the supportedProperties() method
    static PyObject * staticCallback_supportedProperties (PyObject *self, PyObject *args);
    /// implementer for the supportedProperties() method
    PyObject*  supportedProperties(PyObject *args);
    //@}

    /// getter method for special attributes (e.g. dynamic ones)
    PyObject *getCustomAttributes(const char* attr) const;
    /// setter for special attributes (e.g. dynamic ones)
    int setCustomAttributes(const char* attr, PyObject *obj);
    PyObject *_getattr(char *attr);              // __getattr__ function
    int _setattr(char *attr, PyObject *value);        // __setattr__ function

protected:
    std::map<std::string, PyObject*> dyn_methods;

private:
};

} //namespace App

#include "FeaturePythonPyImp.inl" // Type structure of FeaturePythonPyT
/// Methods structure of FeaturePythonPyT
template<class FeaturePyT>
PyMethodDef FeaturePythonPyT<FeaturePyT>::Methods[] = {
    ...

template <class FeatureT>
class FeaturePythonT : public FeatureT
{
    PROPERTY_HEADER(App::FeaturePythonT<FeatureT>);
    ...

protected:
    virtual void onBeforeChange(const Property* prop) {
        FeatureT::onBeforeChange(prop);
        imp->onBeforeChange(prop);
    }
    virtual void onChanged(const Property* prop) {
        imp->onChanged(prop);
        FeatureT::onChanged(prop);
    }

private:
    FeaturePythonImp* imp;
    DynamicProperty* props;
    PropertyPythonObject Proxy;
};
```

3.6.7 Extension framework

This framework is added in 0.17, see feature announcement in forum discussion: Developer Feature: Extensions

In FreeCAD normally inheritance is a chain, it is not possible to use multiple inheritance.

https://github.com/FreeCAD/FreeCAD/blob/master/src/App/ExtensionContainer.h#L36 ***********************************

3.7 Startup process of FreeCADCmd

3.7.1 skeleton of main() function in src/Main/MainCmd.cpp

This cpp script will be compiled into the executable `freecadcmd`, which drop you to a python interpreter with FreeCAD modules path appended to sys.path.

```
    main()
    {
        try {
            // Init phase ===============================
            // sets the default run mode for FC, starts with command prompt
                //if not overridden in InitConfig...
            App::Application::Config()["RunMode"] = "Exit";

            // Inits the Application
            App::Application::init(argc,argv);
        }

        // Run phase =================================
        App::Application::runApplication();


        // Destruction phase ==============================
        Console().Log("FreeCAD terminating...\n");

        // close open documents
        App::GetApplication().closeAllDocuments();

        // cleans up
        Application::destruct();

        Console().Log("FreeCAD completely terminated\n");

        return 0;
    }
```

3.7.2 src/Main/MainPy.cpp

This cpp file will be compiled into a python module “FreeCAD” that can be imported into standard python.

This source code deal with different OS platforms, python 2 or 3 version, by conditional C macro. Set IO rediction for error, output and log.

```
PyMOD_INIT_FUNC(FreeCAD)
//void MainExport initFreeCAD()  // in version 0.16 the function name was called
{
    // Init phase ================
    App::Application::Config()["ExeName"] = "FreeCAD";
    // ...
    // load shared dll/so
    App::Application::init(argc,argv);
}
```

At the end of this function, module is return, see

```
#if PY_MAJOR_VERSION >= 3
    //PyObject* module = _PyImport_FindBuiltin("FreeCAD");
    PyObject* modules = PyImport_GetModuleDict();
    PyObject* module = PyDict_GetItemString(modules, "FreeCAD");
    if (!module) {
        PyErr_SetString(PyExc_ImportError, "Failed to load FreeCAD module!");
    }
    return module;
#endif
```

3.7.3 App::Application class

//singleton pointer to Application is decleared in *Application.cpp* file Application * Application::_pcSingleton = 0; //static member variable

```
void Application::init(int argc, char ** argv)  //static
{

// 1) setup signal handler

initTypes(); // 2) see later sourc code

initConfig(int argc, char ** argv) //std::map<std::string, std::string>
// 3) Environmental variable; LoadParameters();

initApplication(); //4) see below
}

void Application::initTypes(void)  //static
{
    // Base types
    Base::Type                      ::init();
    Base::BaseClass                 ::init();
    ... all other types

void Application::initApplication(void)  //static
{
    // interpreter and Init script ===============================
    // register scripts
    new ScriptProducer( "FreeCADInit",    FreeCADInit    );
    new ScriptProducer( "FreeCADTest",    FreeCADTest    );

    // creating the application
    if (!(mConfig["Verbose"] == "Strict")) Console().Log("Create Application\n");
    Application::_pcSingleton = new Application(0,0,mConfig);

    // set up Unit system default
    ParameterGrp::handle hGrp = App::GetApplication().GetParameterGroupByPath
       ("User parameter:BaseApp/Preferences/Units");
    UnitsApi::setSchema((UnitSystem)hGrp->GetInt("UserSchema",0));

#if defined (_DEBUG)
    Console().Log("Application is built with debug information\n");
#endif

    // starting the init script
    Console().Log("Run App init script\n");
    Interpreter().runString(Base::ScriptFactory().ProduceScript("FreeCADInit"));
}
```

3.7.4 How Python interpreter is integrated

```
Application::Application(ParameterManager * /*pcSysParamMngr*/,
                         ParameterManager * /*pcUserParamMngr*/,
                         std::map<std::string,std::string> &mConfig)
    ://_pcSysParamMngr(pcSysParamMngr),
    //_pcUserParamMngr(pcUserParamMngr),
    _mConfig(mConfig),
    _pActiveDoc(0)
{
    //_hApp = new ApplicationOCC;
    mpcPramManager["System parameter"] = _pcSysParamMngr;
    mpcPramManager["User parameter"] = _pcUserParamMngr;


    // setting up Python binding
    Base::PyGILStateLocker lock;
    PyObject* pAppModule = Py_InitModule3("FreeCAD", Application::Methods, FreeCAD_doc);
    Py::Module(pAppModule).setAttr(std::string("ActiveDocument"),Py::None());

    PyObject* pConsoleModule = Py_InitModule3("__FreeCADConsole__", ConsoleSingleton::Methods, Console_doc);

    // introducing additional classes

    // NOTE: To finish the initialization of our own type objects we must
    // call PyType_Ready, otherwise we run into a segmentation fault, later on.
    // This function is responsible for adding inherited slots from a type's base class.

    //... more code not shown!!!

}

static void Application::runApplication()
{
    // process all files given through command line interface
    processCmdLineFiles();

    if (mConfig["RunMode"] == "Cmd") {
        // Run the comandline interface
        Interpreter().runCommandLine("FreeCAD Console mode");
    }
    else if (mConfig["RunMode"] == "Internal") {
        // run internal script
        Console().Log("Running internal script:\n");
        Interpreter().runString(Base::ScriptFactory().ProduceScript(mConfig["ScriptFileName"].c_str()));
    }
    else if (mConfig["RunMode"] == "Exit") {
        // getting out
        Console().Log("Exiting on purpose\n");
    }
    else {
        Console().Log("Unknown Run mode (%d) in main()?!?\n\n",mConfig["RunMode"].c_str());
    }
}
```

3.8 FreeCADGui start up process

3.8.1 main() in src/Main/MainGui.cpp

This source will be compiled into the program “freecad”, which will take you FreeCAD GUI.

This main function is similar with src/Main/MainCmd.cpp, except it supports both Gui and nonGui mode *App::Application::init(argc, argv);* and *App::Application::destruct();* are still called!

QCoreApplication is defined for WIN32, see src/Main/MainGui.cpp, text banner is defined here

```
main()
{
    App::Application::init(argc, argv);
    Gui::Application::initApplication(); // extra InitApplication();
    // Only if 'RunMode' is set to 'Gui' do the replacement
    if (App::Application::Config()["RunMode"] == "Gui")
            Base::Interpreter().replaceStdOutput();

    try {
        if (App::Application::Config()["RunMode"] == "Gui")
            Gui::Application::runApplication();
        else
            App::Application::runApplication();
    }
    ...
    App::Application::destruct();
}
```

3.8.2 runApplication() in src/Gui/Application.cpp

Constructor of Gui::Application: setting up Python binding

```
/** Override QCoreApplication::notify() to fetch exceptions in Qt widgets
 * properly that are not handled in the event handler or slot.
 */
class GUIApplication : public GUIApplicationNativeEventAware

void Application::runApplication(void)
{
    GUIApplication mainApp(argc, App::Application::GetARGV(), systemExit);
    // set application icon and window title
    const std::map<std::string,std::string>& cfg = App::Application::Config();
    ...
    QCoreApplication::addLibraryPath(plugin);
    ...//setup config, style sheet
    Application app(true); // it is worth of going through the constructor of Gui::Application
    MainWindow mw;
    mw.setWindowTitle(mainApp.applicationName());

    // init the Inventor subsystem
    SoDB::init();
    SIM::Coin3D::Quarter::Quarter::init();
    SoFCDB::init();

    // running the GUI init script
    try {
        Base::Console().Log("Run Gui init script\n");
        Base::Interpreter().runString(Base::ScriptFactory().ProduceScript("FreeCADGuiInit"));
    }
    catch (const Base::Exception& e) {
        Base::Console().Error("Error in FreeCADGuiInit.py: %s\n", e.what());
        mw.stopSplasher();
        throw;
    }
    // stop splash screen and set immediately the active window that may be of interest
    // for scripts using Python binding for Qt
    mw.stopSplasher();
    mainApp.setActiveWindow(&mw);
    ...
    app.activateWorkbench(start.c_str());
    ...
    // run the Application event loop
    Base::Console().Log("Init: Entering event loop\n");
    try {
        std::stringstream s;
        s << App::Application::getTempPath() << App::GetApplication().getExecutableName()
          << "_" << QCoreApplication::applicationPid() << ".lock";
        // open a lock file with the PID
        Base::FileInfo fi(s.str());
        Base::ofstream lock(fi);
        boost::interprocess::file_lock flock(s.str().c_str());
        flock.lock();

        int ret = mainApp.exec();
        if (ret == systemExit)
            throw Base::SystemExitException();

        // close the lock file, in case of a crash we can see the existing lock file
        // on the next restart and try to repair the documents, if needed.
        flock.unlock();
        lock.close();
        fi.deleteFile();
    }

}
```

3.8.3 src/Main/FreeCADGuiPy.cpp

This cpp file will be compiled into a python module “FreeCADGui” that can be imported into standard python.

refer to src/Gui/Application.cpp for details of FreeCAD start up with GUI

It defines the GuiThread class

```
struct PyMethodDef FreeCADGui_methods[] = {
    {"showMainWindow",FreeCADGui_showMainWindow,METH_VARARGS,
     "showMainWindow() -- Show the main window\n"
     "If no main window does exist one gets created"},
    {"exec_loop",FreeCADGui_exec_loop,METH_VARARGS,
     "exec_loop() -- Starts the event loop\n"
     "Note: this will block the call until the event loop has terminated"},
    {"setupWithoutGUI",FreeCADGui_setupWithoutGUI,METH_VARARGS,
     "setupWithoutGUI() -- Uses this module without starting\n"
     "an event loop or showing up any GUI\n"},
    {"embedToWindow",FreeCADGui_embedToWindow,METH_VARARGS,
     "embedToWindow() -- Embeds the main window into another window\n"},
    {NULL, NULL}  /* sentinel */
};


PyMODINIT_FUNC initFreeCADGui()
{
    try {
        Base::Interpreter().loadModule("FreeCAD");
        App::Application::Config()["AppIcon"] = "freecad";
        App::Application::Config()["SplashScreen"] = "freecadsplash";
        App::Application::Config()["CopyrightInfo"] = "\xc2\xa9 Juergen Riegel, Werner Mayer, Yorik van Havre 
        Gui::Application::initApplication();
        Py_InitModule("FreeCADGui", FreeCADGui_methods);
    }
    catch (const Base::Exception& e) {
        PyErr_Format(PyExc_ImportError, "%s\n", e.what());
    }
    catch (...) {
        PyErr_SetString(PyExc_ImportError, "Unknown runtime error occurred");
    }
}
```

Chapter 4

Overview of Gui module

4.1 List of header files in Gui folder

- Application.h Gui related init code, run after `App::Application::initApplication()`

  `Gui::Application` is different from `App::Application`, it mainly deals with Gui stuff, Documents, Views and Workbenches - type system: `initTypes(), initApplication() and runApplication();` - document file open: `importFrom();` - singleton: `*Application::Instance*` - `Gui::Document* activeDocument(void) const;` - `void attachView(Gui::BaseView* pcView);` - `bool activateWorkbench(const char* name);` - `Gui::MacroManager *macroManager(void);` - `Gui::CommandManager &commandManager(void);`

- ApplicationPy.cpp Export Gui::Application methods as FreeCADGui python module

  other ClassNamePy.cpp are also init and incorporated into FreeCADGui.py, Control, Selection module

- FreeCADGuiInit.py function like Init.py and InitGui.py in other module

  define Workbench and StdWorkbech python class, InitApplications(), and add types

Gui components

- Workbench.h class in FreeCAD, each module has one class derived from this

  StdWorkbech <- Workbench <- BaseClass The PythonBaseWorkbench class allows the manipulation of the workbench from Python. `virtual void setupContextMenu(const char* recipient, MenuItem*) const;` The workbench defines which GUI elements (such as toolbars, menus, dockable windows, …) are added to the mainwindow and which gets removed or hidden. To create workbenches you should use the API of WorkbenchManager. [Workbench.cpp, Workbench.h, WorkbenchPyImp.cpp WorkbenchFactory.cpp, WorkbenchManager.cpp, WorkbenchPy.xml WorkbenchFactory.h, WorkbenchManager.h PythonWorkbenchPyImp.cpp, PythonWorkbenchPy.xml]

- Window.h Adapter class to the parameter of FreeCAD for all windows

  Retrieve the parameter group of the specific window by the windowname. `class GuiExport WindowParameter : public ParameterGrp::ObserverType`

- MainWindow.h QMainWindow, also defined MDITabbar class
- GuiConsole.h what is the reationship with PythonConsole?
- PythonEditor.h python macro file view and edit
- PythonConsole.h Where python command can be typed in, GUI commands show up
- PrefWidgets.h The preference widget classes like PrefRadioButton used in *Preference Page*

  PrefRadioButton is derived from QRadioButton and PrefWidget If you want to extend a QWidget class to save/restore its data you just have to derive from this class and implement the methods restorePreferences() and savePreferences().

- StatusWidget.h

Singleton Gui services

- MenuManager.h module can add new Mune and MenutItem
- ToolBarManager.h module can add new ToolBar and ToolBox

- ToolBoxManager.h add ToolBox to MainWindow
- WorkbenchManager.h activate workbench
- DockWindowManager.h

Model-Document-View design pattern

- Document.h Document class's corresponding object in the Gui namespace

  `Gui::Document class` includes a member of `App::Document  class` Its main responsibility is keeping track off open windows for a document and warning on unsaved closes. All handled views on the document must inherit from MDIView

- DocumentModel.h derived from QAbstractItemModel, represents DocumentObject in diff view

  Qt Model-View design to split data and GUI rendering widgets

- View.h define BaseView for various derived *View* class

  DockWindow and MDIView are derived from BaseView, see doxygen inheritance graph module developers need not know such low level API as in [CombiView.cpp, GraphvizView.cpp, ProjectView.h, TreeView.cpp CombiView.h, GraphvizView.h, PropertyView.cpp, TreeView.h CommandView.cpp, HelpView.cpp, PropertyView.h, View.cpp DlgReportView.ui, HelpView.h, ReportView.cpp, View.h DlgSettings3DView.ui, MDIView.cpp, ReportView.h EditorView.cpp, MDIView.h, SelectionView.cpp EditorView.h, ProjectView.cpp, SelectionView.h]

- MDIView.h View binding with `Gui::Document`

  3D view scene is derived from this class, MDIView can be organised by Tab

- DockWindow.h organise diff dockable widgets in workbench

  derived from `BaseView` and `QWidget`

- CombiView.h TreeView+TaskView of the group of `DocumentObject`

  Derived from DockWindows, showDialog(), getTaskPanel()

- PropertyView.h show in CombiView, can modify DocumentObject by setting property

  `class PropertyView : public QWidget, public Gui::SelectionObserver`

Transation, Command, Macro record framework

- Macro.h Collection of python code can be play back
- Command.h Base class for command used in transactonal operation to document

  There are a lot `stdCmd*` classed, CommandManager [Command.cpp, Command.h, CommandTest.cpp, DlgCommandsImp.cpp CommandDoc.cpp, CommandMacro.cpp, CommandView.cpp, DlgCommandsImp.h CommandFeat.cpp, CommandStd.cpp, CommandWindow.cpp, DlgCommands.ui]

- Action.h The Action class is the link between Qt's QAction class and FreeCAD's command classes

  The ActionGroup class is the link between Qt's QActionGroup class and FreeCAD's command classes WorkbenchGroup, WorkbenchComboBox, why defined in this header? UndoAction, RedoAction, ToolboxAction, `class GuiExport WindowAction : public ActionGroup` The RecentFilesAction class holds a menu listed with the recent files.

- ActionFunction.h

Selection in View and identify it DocumentObject tree

- Selection.h represent selected DocumentObject

  see details in *Selection Framework* section [lex.SelectionFilter.c, SelectionFilter.y, SoFCSelectionAction.cpp MouseSelection.cpp, Selection.h, SoFCSelectionAction.h MouseSelection.h, SelectionObject.cpp, SoFCSelection.cpp Selection.cpp, SelectionObject.h, SoFCSelection.h SelectionFilter.cpp, SelectionObjectPyImp.cpp, SoFCUnifiedSelection.cpp SelectionFilter.h, SelectionObjectPy.xml, SoFCUnifiedSelection.h SelectionFilter.l, SelectionView.cpp, SelectionFilter.tab.c, SelectionView.h]

TaskView Framework

- Control.h ControlSingleton is TaskView controller, update all views for document change
- TaskView TaskView is feature setup dialog embedded in left pane

Python related classes

- PythonConsole.h Interactive Python console in dockable windows
- PythonEditor.h QTextEdit with Python grammar highligher
- PythonDebugger.h ???

Widgets with quantity/expression support

- SpinBox.h ExpressionBinding+QSpinBox

  [InputField.h, InputVector.h]

- QuantitySpinBox.h QSpinBox with unit support
- PropertyPage.h PreferencePage and PropertyPage
- TextEdit.h Text input widget

Utility classes

- Thumbnail.h show thumbnail in file explorer
- Splashscreen.h customize FreeCAD startup Splashscreen
- CallTips.h

- WhatsThis.h gives tip for ToolBar for mouse-over event
- Assistant.h `startAssistant(); in QProcess`
- WaitCursor.h hint user to wait and disable user input
- ProgressDialog.h show progress in dialog
- ProgressBar.h show progress in statusbar
- Placement.h derived from Gui::LocationDialog to edit ViewProvider's Placement
- Transform.h derived from Gui::LocationDialog to edit Transormation
- Utilities.h Utility functions
- Flag.h ???

ViewProvider framework, 2D/3D visualization related classes

- ViewProvider.h base class for DocumentObject in rendering

  derived classes: [ViewProviderAnnotation.cpp, ViewProviderInventorObject.cpp ViewProviderAnnotation.h, ViewProviderInventorObject.h ViewProviderBuilder.cpp, ViewProviderMaterialObject.cpp ViewProviderBuilder.h, ViewProviderMaterialObject.h ViewProvider.cpp, ViewProviderMeasureDistance.cpp ViewProviderDocumentObject.cpp, ViewProviderMeasureDistance.h ViewProviderDocumentObjectGroup.cpp, ViewProviderPlacement.cpp ViewProviderDocumentObjectGroup.h, ViewProviderPlacement.h ViewProviderDocumentObject.h, ViewProviderPlane.cpp ViewProviderDocumentObjectPyImp.cpp, ViewProviderPlane.h ViewProviderDocumentObjectPy.xml, ViewProviderPyImp.cpp ViewProviderExtern.cpp, ViewProviderPythonFeature.cpp ViewProviderExtern.h, ViewProviderPythonFeature.h ViewProviderFeature.cpp, ViewProviderPythonFeaturePyImp.cpp ViewProviderFeature.h, ViewProviderPythonFeaturePy.xml ViewProviderGeometryObject.cpp, ViewProviderPy.xml ViewProviderGeometryObject.h, ViewProviderVRMLObject.cpp ViewProvider.h, ViewProviderVRMLObject.h]

- ViewProviderDocumentObject.h base class for view providers of attached document object

  `void attach (App::DocumentObject *pcObject)`, redraw after changing obj's property`updateData (const App::Property *)`

- ViewProviderGeometryObject.h base class for all view providers that display geometric data, like mesh, point cloudes and shapes
- ViewProviderExtern.h render OpenInventor *.iv file or iv string
- ViewProviderAnnotation.h Text render in 3D scene
- ViewProviderFeature.h has full Python support on this class
- ViewProviderPythonFeature.h ???

OpenInventor/Coin3D rendering related classes

- SoFCDB.h The FreeCAD database class to initializee all onw Inventor nodes
- SoFCSelection.h extend SoNode of Coin3D/OpenInventor

  header file name begins with *SoFC* is derived from OpenInventor objects used by FreeCAD

- View3DInventor.h contains View3DInventorViewer obj and control parameter and event

  `View3DInventor : public MDIView, public ParameterGrp::ObserverType`

- View3DInventorViewer.h 3D rendering in QGraphicsView

  bridge the gap between OpenInvetorObject and ViewProvider derived from Quarter::SoQTQuarterAdaptor and Gui::SelectionSingleton::ObserverType

- View3DPy.h PyObject controls *View3DInventor*, like view angle, viewLeft()…

  [View3DInventor.cpp, View3DInventorRiftViewer.cpp, View3DPy.cpp View3DInventorExamples.cpp, View3DInventorRiftViewer.h, View3DPy.h View3DInventorExamples.h, View3DInventorViewer.cpp, View3DViewerPy.cpp View3DInventor.h, View3DInventorViewer.h, View3DViewerPy.h]

Network related related classes

- DownloadItem.h

- DownloadManager.h

- NetworkRetriever.h

subfolders in Gui

- 3Dconnexion 3D mouse 3Dconnexion's supporting lib
- Inventor Inventor 3D rendering lib
- TaskView TaskView Framework for FreeCAD Gui
- QSint Collection of extra Qt widgets from community
- iisTaskPanel Task panel UI widgets, now part of QSint
- propertyeditor Widget for property edit for DocumentObject
- Language translation for FreeCADGui
- Icons icon for commands

4.2 Important classes in Gui namespace

4.2.1 Gui::Application

```
Gui::Application::Instance->activeDocument()
```

4.2.2 Gui::Document

`Gui::Document()` includes `App::Document` but not inherits from it!

```
class GuiExport Document : public Base::Persistence
{
public:
    Document(App::Document* pcDocument, Application * app);
```

4.2.3 GUI components

http://iesensor.com/FreeCADDoc/0.16-dev/df/d3c/classGui_1_1BaseView.html

PropertyView has PropertyEditor,

```
class PropertyView : public QWidget, public Gui::SelectionObserver
class PropertyDockView : public Gui::DockWindow
```

4.2.4 Gui Services API

Frequently included headers

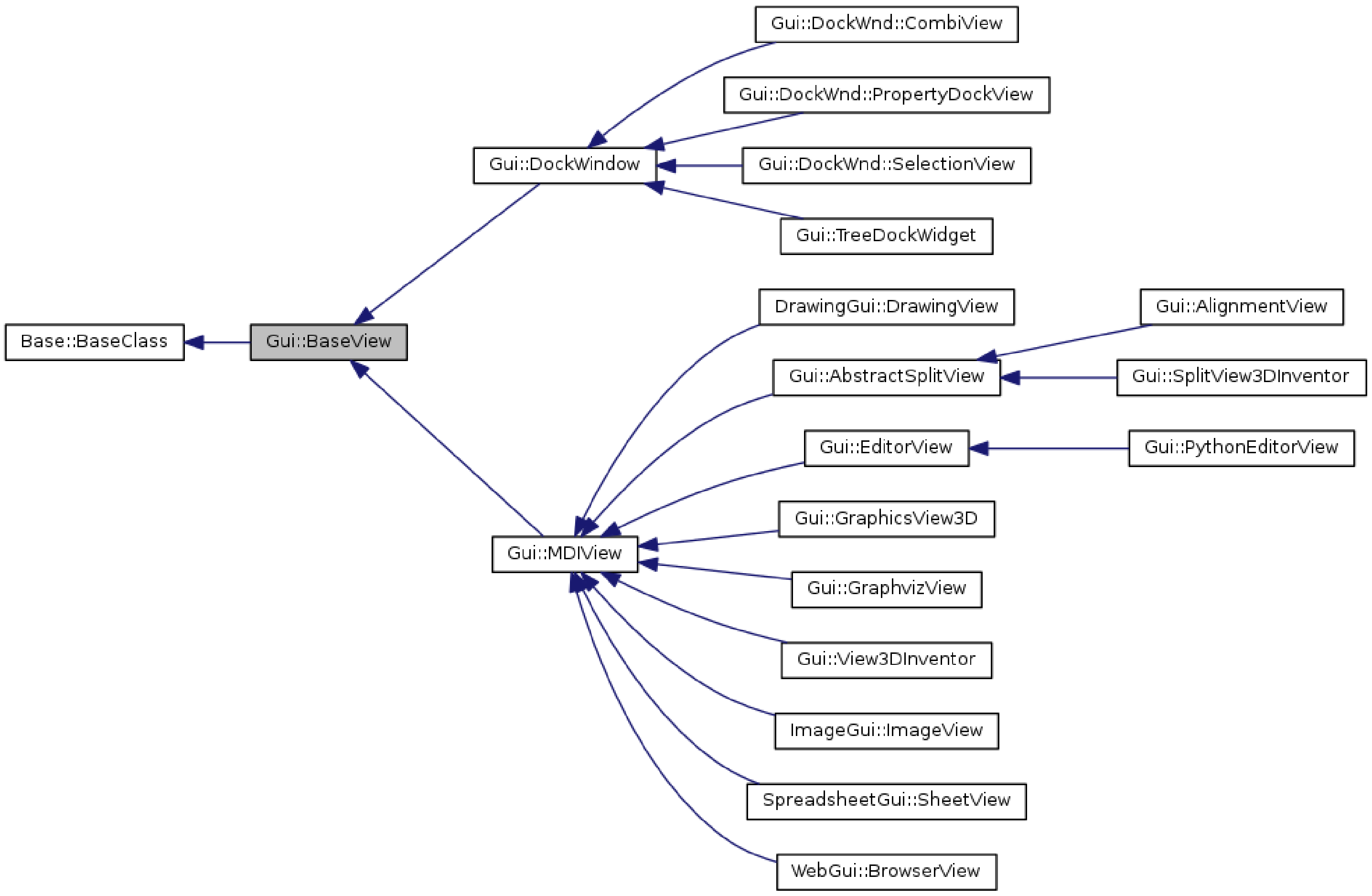


Figure 4.1: BaseView inheritance graph

```
<Application.h>   //Open and close Document files, control multiple documents
<Selection.h>     //whenever selection in 3D scene is needed
<Command.h>       //scripting, undo support
<CombiView.h>     //object hierarchy true view
<MainWindow.h>    //
<Control.h>       //
```

C++ has different API with python API `FreeCADGui.getCommandManager()`, e.g. `CommandManager &mgr = Gui::Application::Ins`

4.2.5 Selection API in Python and C++

The key C++ API are defined in src/Gui/Selection.h. see the doxygen document: http://free-cad.sourceforge.net/SrcDocu/d4/dca/classGui_1_1SelectionSingleton.html

The `SelectionSingleton` keeps track of the selection state of the whole application. It gets messages from all entities which can alter the selection (e.g. tree view and 3D-view) and sends messages to entities which need to keep track on the selection state. `SelectionObserver` class implements the observer pattern; `SelectionGate` class enables the selective pickup for the specific type (filtering).

The selection consists mainly out of following information per selected object: - document (pointer) - Object (pointer) - list of subelements (list of strings) - 3D coordinates where the user clicks to select (Vector3d)

Also the preselection is managed. That means you can add a filter to prevent selection of unwanted objects or subelements.

src/Gui/Selection.h

```
std::vector<App::DocumentObject*> getObjectsOfType(const char* typeName, const char* pDocName=0) const;

    struct SelObj {
        const char* DocName;
        const char* FeatName;
        const char* SubName;
        const char* TypeName;
        App::Document* pDoc;
        App::DocumentObject*  pObject;
        float x,y,z;
    };

/** Returns a vector of SelObj struct
* If no document name is given the objects of the active are returned.
* If nothing for this Document is selected an empty vector is returned.
* The vector reflects the sequence of selection.
*/
     std::vector<SelObj> getSelection(const char* pDocName=0) const;

    bool hasSelection() const;
    bool hasSelection(const char* doc) const;

/** Returns a vector of SelectionObjects
* If no document name is given the objects of the active are returned.
* If nothing for this document is selected an empty vector is returned.
* The vector reflects the sequence of selection.
*/
     std::vector<Gui::SelectionObject> getSelectionEx(const char* pDocName=0,Base::Type typeId=App::DocumentOb
```

[python Selection API](https://www.freecadweb.org/wiki/Selection_API>

Python API is a wrapper of methods of `SelectionSingleton` in src/Gui/Selection.h, i.e. `addSelection(), clearSelection, removeSelection(FreCAD.Object), isSelcted() getSelection()` return a list of `DocumentObject*` for the current document or specified document. Meanwhile, `getSelectionEx()` return a list of `SelectionObject`, which mainly designed for subfeature selection. Notably,`Struct SelObj` has the basic information, but it not exposed to Python.

src/Gui/SelectionObject.h

```
class GuiExport SelectionObject : public Base::BaseClass

    /// get the SubElement name of this SelectionObject
    inline const std::vector<std::string> &getSubNames(void) const { return SubNames; }
    /// are there any SubNames selected
    bool hasSubNames(void)const { return SubNames.size() != 0; }
    /// get the name of the Document of this SelctionObject
    inline const char* getDocName(void) const { return DocName.c_str(); }
    /// get the name of the Document Object of this SelectionObject
    inline const char* getFeatName(void) const { return FeatName.c_str(); }
    /// get the Type of the selected Object
    inline const char* getTypeName(void) const { return TypeName.c_str(); }
    /// get the selection points
    inline const std::vector<Base::Vector3d> getPickedPoints(void) const { return SelPoses; }

    /// returns the selected DocumentObject or NULL if the object is already deleted
    const App::DocumentObject *getObject(void) const;
    /// returns the selected DocumentObject or NULL if the object is already deleted
    App::DocumentObject *getObject(void);

    /// check the selected object is a special type or derived of
    bool isObjectTypeOf(const Base::Type& typeId) const;

    /// returns python expreasion sutably for assigning to a LinkSub property
    std::string getAsPropertyLinkSubString(void) const;
```

4.3 ViewProvider framework and 3D rederring

See the `ViewProvider` inheritance graph and collaboration diagram, http://iesensor.com/FreeCADDoc/0.16-dev/db/d77/classGui_1_1ViewProviderGeometryObject.html

4.3.1 `Gui::ViewProvider`

The general interface for all visual stuff in FreeCAD. This class is used to generate and handle all around visualizing and presenting objects from the FreeCAD App layer to the user. This class and its descendants have to be implemented for any object type in order to show them in the 3DView and TreeView.

Inventor object will be created and ref in the constructor if defined in this base class; while in destructor, `pyViewObject.unref()` is called, in addition to unref open inventor objects. `show()` and `hide()` are virtual functions, but they have implementation, intensive implementation happens in `Gui::DocumentObjectViewProvider`. `PyObject* ViewProvider::getPyObject()` has its implementation so all the derived classes for the specific python type, however, only one PyObject destruction happens in this based class (`pyViewObject.unref()` is called).

Base class `ViewProvider`, derived from `PropertyContainer` , is surprisingly short in coding; the derived classes have implementation. Some important methods for python module developer are listed: - Object returns the DocumentObject this ViewProvider is associated to - RootNode returns the Root coin node of this object - toString() returns a string representation of the coin node of this object - update() this method is executed whenever any of the properties of this ViewProvider changes

see details python API manual at https://www.freecadweb.org/api/ViewProvider.html

grouped in doxygen document

OpenInventor related objects () are declared as protected var:

```
SoSeparator *   pcAnnotation // The root separator for annotations.
```

```
SoSwitch *  pcModeSwitch  // this is the mode switch, all the different viewing modes are collected here
```

```
SoSeparator *   pcRoot  // the root Separator of the ViewProvider.
```

```
SoTransform *   pcTransform //  this is transformation for the provider
```

src/Gui/ViewProvider.h

```
void ViewProvider::update(const App::Property* prop)
{
    // Hide the object temporarily to speed up the update
    if (!isUpdatesEnabled())
        return;
    bool vis = ViewProvider::isShow();
    if (vis) ViewProvider::hide();
    updateData(prop);
    if (vis) ViewProvider::show();
}

/// Reimplemented from subclass
void ViewProvider::onChanged(const App::Property* prop)
{
    Application::Instance->signalChangedObject(*this, *prop);
}
```

If Open Inventor objects are defined as property, `attach()` needs to be overridden.

4.3.2 Gui::DocumentObjectViewProvider

This is the counterpart of the `DocumentObject` in the GUI space. It is only present when FreeCAD runs in GUI mode (e.g. show(), hide(), update()). It contains all that is needed to represent the `DocumentObject` in the 3D view and the FreeCAD CombiView. It implements `show() hide() attach()`, also restores view provider from document file loaded: `virtual void finishRestoring ()` and `virtual void startRestoring ()`.

This class has detailed doxygen code documentation in this header file, Similar with the `ViewProvider` class, `show() hide()` are virtual member functions but with implementation.

src/Gui/ViewProviderDocumentObject.cpp This class defines two new Properties in constructor.

```
    ADD_PROPERTY(DisplayMode,((long)0));
    ADD_PROPERTY(Visibility,(true));
```

Thereby, `onChanged(const App::Property* prop)` is reimplemented

```
void ViewProviderDocumentObject::onChanged(const App::Property* prop)
{
    if (prop == &DisplayMode) {
        setActiveMode();
    }
    else if (prop == &Visibility) {
        // use this bit to check whether show() or hide() must be called
        if (Visibility.testStatus(App::Property::User2) == false) {
            Visibility.setStatus(App::Property::User2, true);
            Visibility.getValue() ? show() : hide();
            Visibility.setStatus(App::Property::User2, false);
        }
    }

    ViewProvider::onChanged(prop);
}
```

DisplayMode related code is found in`attach()`

```
Gui::MDIView* ViewProviderDocumentObject::getActiveView() const
viewer->getSoRenderManager()->getViewportRegion());
viewer->getSoRenderManager()->getCamera();
```

Similar with ViewProvider class, `show() hide()` are virtual member functions but with implementation.

```
void ViewProviderDocumentObject::updateView()
{
    std::map<std::string, App::Property*> Map;
    pcObject->getPropertyMap(Map);

    // Hide the object temporarily to speed up the update
    bool vis = ViewProvider::isShow();
    if (vis) ViewProvider::hide();
    for (std::map<std::string, App::Property*>::iterator it = Map.begin(); it != Map.end(); ++it) {
        updateData(it->second);
    }
    if (vis) ViewProvider::show();
}
```

4.3.3 `Gui::ViewProviderGeometryObject`

The base class for all view providers that display geometric data, like mesh, point clouds, and shapes. `drag, select(pick), boundingbox, sensorCallback()`

src/Gui/ViewProviderGeometryObject.cpp

```
    ADD_PROPERTY(ShapeColor,(r, g, b));
    ADD_PROPERTY(Transparency,(0));
    Transparency.setConstraints(&intPercent);
    App::Material mat(App::Material::DEFAULT);
    ADD_PROPERTY(ShapeMaterial,(mat));
    ADD_PROPERTY(BoundingBox,(false));
    ADD_PROPERTY(Selectable,(true));
```

`void ViewProviderGeometryObject::onChanged(const App::Property* prop)` just call parent methods, in addition to properties defined in this class. `void ViewProviderGeometryObject::updateData(const App::Property* prop)`, update Placement and PropertyComplexGeoData.

`Gui::ViewProviderBuilder`: Render complex geometry like points.

4.3.4 `Fem::ViewProviderFemConstraint`

This class draws some visual objects, arrows and cubes in 3D view, see src/Mod/Fem/Gui/ViewProviderFemConstraint.cpp

- Some more inventor objects are created in Constructor: `cpp SoPickStyle* ps = new SoPickStyle(); ps->style = SoPickStyle::UNPICKABLE;`
- `unsetEdit()` is shared by all derived classes for TaskPanel.
- `onChange()` for updated drawing for changed ViewProvider properties `void ViewProviderFemConstraint::onChanged(const App::Property* prop) { if (prop == &Mirror || prop == &DistFactor) {     updateData(prop);`

src/Mod/Fem/Gui/ViewProviderFemConstraintFluidBoundary.cpp

Draw 3D objects more specifically for different constraint types

- `bool ViewProviderFemConstraintFluidBoundary::setEdit(int ModNum)` activate the taskpanel dialog
- `void ViewProviderFemConstraintFluidBoundary::updateData(const App::Property* prop)` for DocumentObject property update

4.3.5 3D CAD Part rendering

src/Mod/Part/Gui/ViewProvider.h

The base class for all CAD features like boolean operation, fillet, etc... implemented by OpenCASCADE.

```
TopoDS_Shape getShape (const SoPickedPoint *) const
Standard_Boolean computeEdges (SoGroup *root, const TopoDS_Shape &myShape)
Standard_Boolean computeFaces (SoGroup *root, const TopoDS_Shape &myShape, double defl)
Standard_Boolean computeVertices (SoGroup *root, const TopoDS_Shape &myShape)
```

src/Mod/Part/Gui/ViewProviderExt.cpp has concrete code to render OpenCASCADE CAD object in 3D view

```
// this was 0.16dev
class PartGuiExport ViewProviderPart : public ViewProviderPartExt
{
    SoCoordinate3     * coords;
    SoBrepFaceSet     * faceset;
    SoNormal          * norm;
    SoNormalBinding   * normb;
    SoBrepEdgeSet     * lineset;
    SoBrepPointSet * nodeset;
}
class ViewProviderShapeBuilder : public Gui::ViewProviderBuilder
```

In version 0.19,

```
class PartGuiExport ViewProviderPartExt : public Gui::ViewProviderGeometryObject
```

Surprisingly, the filename and class name are not consistent! In version 0.16dev, the cpp file name was ViewProviderPartExt.cpp

src/Mod/Part/Gui/ViewProviderPython.cpp it is possible to access ViewProvider property in Python by aggregation: `typedef Gui::ViewProviderPythonFeatureT<ViewProviderPart> ViewProviderPython;`

src/Mod/Part/Gui/ViewProviderCylinderParametric.cpp `class PartGuiExport ViewProviderCylinderParametric:public ViewProviderPart`

src/Mod/Part/Gui/DlgPartCylinderImp.cpp no concrete code

src/Mod/Part/App/FeaturePartBox.h

```
/** App::Feature: Base class of all shape feature classes in FreeCAD  */
class PartExport Feature : public App::GeoFeature

class PartExport Primitive : public Part::AttachableObject

class PartExport Box :public Part::Primitive
App::DocumentObjectExecReturn *Box::execute(void)
{
    double L = Length.getValue();
    double W = Width.getValue();
    double H = Height.getValue();

    if (L < Precision::Confusion())
        return new App::DocumentObjectExecReturn("Length of box too small");

    if (W < Precision::Confusion())
        return new App::DocumentObjectExecReturn("Width of box too small");

    if (H < Precision::Confusion())
        return new App::DocumentObjectExecReturn("Height of box too small");

    try {
        // Build a box using the dimension attributes
        BRepPrimAPI_MakeBox mkBox(L, W, H);
        TopoDS_Shape ResultShape = mkBox.Shape();
        this->Shape.setValue(ResultShape);
    }
    catch (Standard_Failure) {
        Handle_Standard_Failure e = Standard_Failure::Caught();
        return new App::DocumentObjectExecReturn(e->GetMessageString());
    }

    return App::DocumentObject::StdReturn;
}
```

4.3.6 View3DInventor class

This class derived from Qt MdiView

```
class GuiExport View3DInventor : public MDIView, public ParameterGrp::ObserverType
{
    setOverlayWidget(QWidget*);
    ... mouse  and keyboard events
View3DInventorViewer *getViewer(void) const {return _viewer;}

}
//
class View3DInventorPy : public Py::PythonExtension<View3DInventorPy>
class View3DInventorViewerPy : public Py::PythonExtension<View3DInventorViewerPy>
```

Note: `Quarter::SoQTQuarterAdaptor` is derived from `QGraphicsView`

```
class GuiExport View3DInventorViewer : public Quarter::SoQTQuarterAdaptor, public Gui::SelectionSingleton::Obs
```

```
Gui::MDIView* ViewProviderDocumentObject::getInventorView() const
{
    App::Document* pAppDoc = pcObject->getDocument();
    Gui::Document* pGuiDoc = Gui::Application::Instance->getDocument(pAppDoc);

    Gui::MDIView* mdi = pGuiDoc->getEditingViewOfViewProvider(const_cast<ViewProviderDocumentObject*>(this));
    if (!mdi) {
        mdi = pGuiDoc->getViewOfViewProvider(const_cast<ViewProviderDocumentObject*>(this));
    }

    return mdi;
}

Gui::MDIView* ViewProviderDocumentObject::getActiveView() const
{
    App::Document* pAppDoc = pcObject->getDocument();
    Gui::Document* pGuiDoc = Gui::Application::Instance->getDocument(pAppDoc);
    return pGuiDoc->getActiveView();
}
```

4.3.7 ViewProivder and 3DViewer

The initialization process of the View3DViewer object is highly complex. src/Gui/View3DInventorViewer.cpp

```
init()
{
...
    selectionRoot = new Gui::SoFCUnifiedSelection();
    selectionRoot->applySettings();

    // set the ViewProvider root node
    pcViewProviderRoot = selectionRoot;
...
}
```

adds an ViewProvider to the view, e.g. from a feature

```
void View3DInventorViewer::addViewProvider(ViewProvider* pcProvider)
{
    SoSeparator* root = pcProvider->getRoot();

    if (root) {
        pcViewProviderRoot->addChild(root);
        _ViewProviderMap[root] = pcProvider;
    }

    SoSeparator* fore = pcProvider->getFrontRoot();

    if (fore)
        foregroundroot->addChild(fore);

    SoSeparator* back = pcProvider->getBackRoot();

    if (back)
        backgroundroot->addChild(back);

    pcProvider->setOverrideMode(this->getOverrideMode());
    _ViewProviderSet.insert(pcProvider);
}
```

setSceneGraph(pcViewProviderRoot);

4.3.8 2D drawing rendering using Qt native QGraphicsView

src/Mod/Drawing/Gui/ViewProviderView.cpp Qt2D drawing, not 3D rendering! src/Mod/Drawing/Gui/DrawingView.h

```
class DrawingGuiExport SvgView : public QGraphicsView
class DrawingGuiExport DrawingView : public Gui::MDIView
```

4.3.9 further reading on ViewProvider

see source code analysis in the later chapters: src/Mod/Fem/Gui/ViewProviderResult.cpp

Render 3D object is possbile with pure python, *import pivy*

4.4 selection framework

The SelectionSingleton class keeps track of the selection state of the whole application. For selection in 3D view, selection is based on Inventor classes: “SoPickStyle” “SoPick” and “SoSelection”.

It gets messages (Subject and Observer model) from all entities which can alter the selection (e.g. tree view and 3D-view) and sends messages to entities which need to keep track on the selection state.

4.4.1 src/Gui/Selection.h

This file has defined important classes: SelectionObserver SelectionChanges SelectionObserverPython SelectionGate - SelectionGate: allows or disallows selection of certain types. - SelectionObserver: observer pattern - SelectionChanges: as message for Observer

This file is well documented, see the header file for all API src/Gui/Selection.h

```
class GuiExport SelectionSingleton : public Base::Subject<const SelectionChanges&>

bool SelectionSingleton::setPreselect(const char* pDocName, const char* pObjectName, const char* pSubName, flo
```

4.4.2 src/Gui/SelectionObject.h thin wrapper of DocumentObject pointer

```
class GuiExport SelectionObject : public Base::BaseClass
{

    /// get the name of the Document Object of this SelectionObject
    inline const char* getFeatName(void) const { return FeatName.c_str(); }

    /// returns the selected DocumentObject or NULL if the object is already deleted
    const App::DocumentObject *getObject(void) const;
    ...
```

4.4.3 src/Gui/SelectionView.h show present selection in QListWidget of DockWindow

```
namespace Gui {
namespace DockWnd {

/** A test class. A more elaborate class description.  */
class SelectionView : public Gui::DockWindow,
                      public Gui::SelectionSingleton::ObserverType
{
    Q_OBJECT
    ...
    /// Observer message from the Selection
    virtual void OnChange(Gui::SelectionSingleton::SubjectType &rCaller,
                          Gui::SelectionSingleton::MessageType Reason);


    bool onMsg(const char* pMsg,const char** ppReturn);

    virtual const char *getName(void) const {return "SelectionView";}

    /// get called when the document is changed or updated
    virtual void onUpdate(void);

    QListWidget* selectionView;
```

4.4.4 src/Gui/SelectionFilter.h expression based filtering

This class builds up a type/count tree out of a string to test very fast a selection or object/subelement type against it.

Example strings are: "SELECT Part::Feature SUBELEMENT Edge", "SELECT Robot::RobotObject", "SELECT Robot::RobotObject COUNT 1..5"

4.4.5 src/Gui/MouseSelection.h

4.4.6 Example of getSelection

where is this piece of code?

```
    std::vector<Gui::SelectionObject> selection = getSelection().getSelectionEx();

    if (selection.size() != 1) {
        QMessageBox::warning(Gui::getMainWindow(), QObject::tr("Wrong selection"),
            QObject::tr("Select an edge, face or body. Only one body is allowed."));
        return;
    }
    if (!selection[0].isObjectTypeOf(Part::Feature::getClassTypeId())){
        QMessageBox::warning(Gui::getMainWindow(), QObject::tr("Wrong object type"),
            QObject::tr("Fillet works only on parts"));
        return;
    }
```

4.5 Command framework

Command framework is well-established design pattern, it C++ code sample is listed here, while python side code is much easier. Example code can be found in Fem module in the later chapters.

This section is copied from Doxygen generated document version 0.16dev, accessed: Oct 2015 Doxygen document: Module->Gui->Command Framework

4.5.1 Overview of command framework

In GUI applications many commands can be invoked via a menu item, a toolbar button or an accelerator key. The answer of Qt to master this challenge is the class QAction. A QAction object can be added to a popup menu or a toolbar and keep the state of the menu item and the toolbar button synchronized.

For example, if the user clicks the menu item of a toggle action then the toolbar button gets also pressed and vice versa. For more details refer to your Qt documentation.

4.5.2 Drawbacks of QAction

Since QAction inherits QObject and emits the triggered() signal or toggled() signal for toggle actions it is very convenient to connect these signals e.g. with slots of your MainWindow class. But this means that for every action an appropriate slot of MainWindow is necessary and leads to an inflated MainWindow class. Furthermore, it's simply impossible to provide plugins that may also need special slots – without changing the MainWindow class.

4.5.3 Way out

To solve these problems we have introduced the command framework to decouple QAction and MainWindow. The base classes of the framework are Gui::CommandBase and Gui::Action that represent the link between Qt's QAction world and the FreeCAD's command world.

The Action class holds a pointer to QAction and CommandBase and acts as a mediator and – to save memory – that gets created (Gui::CommandBase::createAction()) not before it is added (Gui::Command::addTo()) to a menu or toolbar.

Now, the implementation of the slots of MainWindow can be done in the method activated() of subclasses of Command instead.

For example, the implementation of the "Open file" command can be done as follows.

```
class OpenCommand : public Command
{
public:
  OpenCommand() : Command("Std_Open")
  {
    // set up menu text, status tip, ...
    sMenuText     = "&Open";
    sToolTipText  = "Open a file";
    sWhatsThis    = "Open a file";
    sStatusTip    = "Open a file";
    sPixmap       = "Open"; // name of a registered pixmap
    sAccel        = "Shift+P"; // or "P" or "P, L" or "Ctrl+X, Ctrl+C" for a sequence
  }
protected:
  void activated(int)
  {
    QString filter ... // make a filter of all supported file formats
    QStringList FileList = QFileDialog::getOpenFileNames( filter,QString::null, getMainWindow() );
    for ( QStringList::Iterator it = FileList.begin(); it != FileList.end(); ++it ) {
      getGuiApplication()->open((*it).latin1());
    }
  }
};
```

An instance of OpenCommand must be created and added to the Gui::CommandManager to make the class known to FreeCAD. To see how menus and toolbars can be built go to the Workbench Framework.

4.5.4 Boost::signal is used

- Boost signal but it is not maintained, how about migration to boost.signal2

  The Boost.Signals2 (Thread-safe) library is an implementation of a managed signals and slots system. Signals represent callbacks with multiple targets, and are also called publishers or events in similar systems. Signals are connected to some set of slots, which are callback receivers (also called event targets or subscribers), which are called when the signal is “emitted.”

4.6 TaskView Framework: UI for interactive design

Both Qt C++ and python (file names start with *TaskPanel*) are used to design the UI (*.ui file generated by QtDesigner) for FreeCAD. Related to `setEdit()`, `unsetEdit()` in `ViewProvider` class. Another Qt library ** is used. An image shows the taskpanel is welcomed here!

4.6.1 Important classed related to TaskView

class export to Python: TaskDialog src/Gui/TaskView/TaskDialogPython.h

src/Gui/TaskView/TaskDialog.h

```
class TaskDialog{
QObject
...
protected: /// List of TaskBoxes of that dialog
std::vector<QWidget*> Content;

}
```

src/Gui/TaskView/TaskView.h

```
class GuiExport TaskGroup : public QSint::ActionBox, public TaskContent
class GuiExport TaskView : public QScrollArea, public Gui::SelectionSingleton::ObserverType
{
//boost::signal connection + slot to App::Document
https://github.com/FreeCAD/FreeCAD/blob/master/src/Gui/TaskView/TaskView.h
// this is an example of QObject event system and boost::signal
}
class GuiExport TaskWatcher : public QObject, public Gui::SelectionFilter
/// List of TaskBoxes of that dialog
std::vector<QWidget*> Content;
```

4.6.2 Controller of TaskView and TaskDialog

```
//break naming convection
class GuiExport ControlSingleton : public QObject  , control Gui::TaskPanel::ControlDialog

/** The control class
 */
class GuiExport ControlSingleton : public QObject
{
     Q_OBJECT

public:
    static ControlSingleton& instance(void);
    static void destruct (void);

    /** @name dialog handling
     *  These methods are used to control the TaskDialog stuff.
     */
    //@{
    /// This method starts a task dialog in the task view
    void showDialog(Gui::TaskView::TaskDialog *dlg);
    Gui::TaskView::TaskDialog* activeDialog() const;
    //void closeDialog();
    //@}

    /** @name task view handling
     */
    //@{
    Gui::TaskView::TaskView* taskPanel() const;
    /// raising the model view
    void showModelView();
    /// get the tab panel
    QTabWidget* tabPanel() const;  //name should be: getTabPanel()
    //@}

    bool isAllowedAlterDocument(void) const;
    bool isAllowedAlterView(void) const;
    bool isAllowedAlterSelection(void) const;

public Q_SLOTS:
    void accept();
    void reject();
    void closeDialog();
    /// raises the task view panel
    void showTaskView();

private Q_SLOTS:
    /// This get called by the TaskView when the Dialog is finished
    void closedDialog();

private:
    Gui::TaskView::TaskView *getTaskPanel();

private:
    struct status {
        std::bitset<32> StatusBits;
    } CurrentStatus;

    std::stack<status> StatusStack;
    Gui::TaskView::TaskDialog *ActiveDialog;

private:
    /// Construction
    ControlSingleton();
    /// Destruction
    virtual ~ControlSingleton();

    static ControlSingleton* _pcSingleton;
};

/// Get the global instance
inline ControlSingleton& Control(void)
{
    return ControlSingleton::instance();
}
```

src/Gui/TaskView

4.6.3 TaskDriver

In module like Fem, src/Mod/Fem/Gui/TaskDriver.h `class TaskDriver : public Gui::TaskView::TaskBox`

multiple ui files are included into src/Mod/Fem/Gui/TaskPostBoxes.h

4.6.4 preference page

src/Gui/PropertyPage.h

`class GuiExport PreferencePage : public QWidget` `class GuiExport PreferenceUiForm : public PreferencePage`

src/Mod/Fem/Gui/DlgSettingsFemCcxImp.h [src/Mod/Fem/Gui/PrefWidgets.h] wideges with save

`class DlgSettingsFemCcxImp : public Gui::Dialog::PreferencePage, public Ui_DlgSettingsFemCcxImp`

4.7 Expression and quantity

4.8 Internationalization with FreeCAD

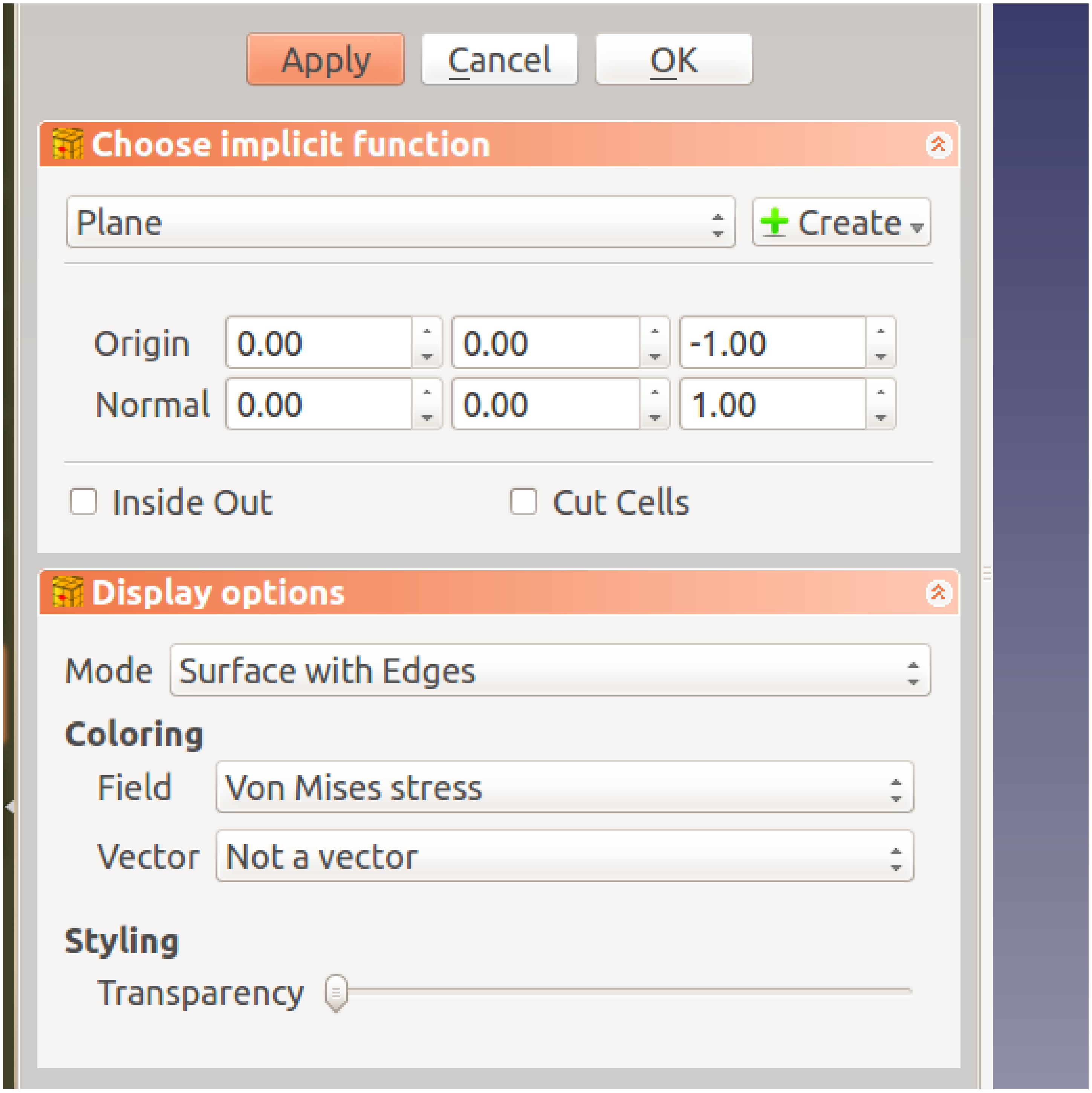
4.8.1 Overview of FreeCAD i18n

This section is mainly copied from FreeCAD documentation, see Internationalization with FreeCAD Doxygen document position: Module->Gui->Internationalization with FreeCAD

The internationalization of FreeCAD makes heavy use of the internationalization support of Qt. For more details refer to your Qt documentation. As FreeCAD will migrated to Qt5 in the future, `QString::fromLatin1()` should be used to convert C-style char array and `std::string` in GUI code.

4.8.2 integrate a new language into FreeCAD

To integrate a new language into FreeCAD or one of its application modules you have to perform the following steps:



Plane

Origin	0.00	0.00	-1.00
Normal	0.00	0.00	1.00

☐ Inside Out ☐ Cut Cells

Mode Surface with Edges

Field Von Mises stress

Vector Not a vector

Transparency

Figure 4.2: mutltile task panel

4.8.2.1 Creation of a .ts file

First you have to generate a .ts file for the language to be translated. You can do this by running the lupdate tool in the bin path of your Qt installation. As argument you can specify either all related source files and the .ts output file or a Qt project file (.pro) which contains all relevant source files.

4.8.2.2 Translation into your language

To translate the english string literals into the language you want to support you can open your .ts file with QtLinguist and translate all literals by hand. Another way for translation is to use the tool tsauto from Sebastien Fricker.This tool uses the engine from Google web page (www.google.com).ts auto supports the languages

To get most of the literals translated you should have removed all special characters (like &, !, ?, . . .). Otherwise the translation could fail. After having translated all literals you can load the .ts file into QtLinguist and invoke the menu item Release which generates the binary .qm file.

4.8.2.3 Integration of the .qm file

The .qm file should now be integrated into the GUI library (either of FreeCAD itself or its application module). The .qm file will be embedded into the resulting binary file. So, at runtime you don't need any .qm files any more. Indeed you will have a bigger binary file but you haven't any troubles concerning missing .qm files.

To integrate the .qm file into the executable you have to create a resource file (.qrc), first. This is an XML file where you can append the .qm file. For the .qrc file you have to define the following curstom build step inside the Visual Studio project file:

Command Line: `rcc.exe -name -o "$(InputDir)qrc_$(InputName).cpp"` Outputs: qrc_.cpp

For the gcc build system you just have to add the line .qrc to the BUILT_SOURCES sources section of the Makefile.am, run automake and configure (or ./confog.status) afterwards.

4.8.2.4 Q_INIT_RESOURCE

Finally, you have to add a the line `Q_INIT_RESOURCE(resource);` where resource is the name of the .qrc file. That's all!

4.8.3 Update of FreeCAD translation

Online translation project: https://crowdin.com/project/freecad

Chapter 5

Introduction to Python Wrapping

It is the python interpreter that makes magic of scripting, macro recording, etc. While wrapping cpp code in python is a tough story.

5.1 Overview of wrapping C++ into Python module

5.1.1 A list of C++ to python wrapping tools

Wrapping binary library is possible, if header files are available.

For C code/lib interfacing

- C-API:`#include <python.h>`
- `ctypes` foreign function library/module from the python standard lib, to wrap binary C library. see cytpes official doc
- cffi: an easier approach than `ctypes`, but support only C
- PyCXX: C++ version for C-API: `<python.h>`

Using C++ template to simplify wrapping

- boost.python: part of boost lib, it avoid lots of boiler-plate code by c++ template technology

see boost doc to build a wrapping module in C++. https://wiki.python.org/moin/boost.python/BuildingExtensions

`pybind11`: similar but with more features than boost.python, using C++11 compiler features.

Automatic binding generation

- cppyy: LLVM JIT solution without writing wrap code, based on `cling`, still in heavy-development
- PyBindGen: similar with cppyy, using GCC to parse C++ header and generate wrapping code.
- `binder`, a tool can generate pybind11 wrapping code automatically
- wrapping: https://github.com/personalrobotics/chimera

Dedicated wrapping tools mainly targeting on the specific project, but can be used in other projects

- Qt5: https://wiki.qt.io/Qt_for_Python/Shiboken
- GTK3 : see architecture of object introspectin, it is based on libFFI.so and typelib (API info)
- VTK, etc.

Other solutions

- swig: wrap C/C++ to several languages like Java, and interpreting lanugages like python, etc.
- cython3: write c++ module in python grammar, it is possible to wrap existing shared binary library given header files.
- SciPy Weave: part of SciPy package, wrap only C code but automatically.
- pyrex: discontinued, superseded by Cython
- py++: automatically extract cpp method and parameter types using gcc-xml and generate the wrapping code

  Disclaimer: This short description to each tool is very subjective, and for no means it is a complete list. the following suggestion is highly subjective

5.1.2 example wrapping code by Pybind11

`pybind11`: latest solution based on C++11 feature, similar but simpler API as boost::python. It is header only and smaller in binary glue code and faster building, see example

example of C++ class wrapping in pybind11

```
#include <pybind11/pybind11.h>

namespace py = pybind11;

PYBIND11_PLUGIN(example) {
    py::module m("example", "pybind11 example plugin");

    py::class_<Pet>(m, "Pet")
        .def(py::init<const std::string &>())
        .def("setName", &Pet::setName)
        .def("getName", &Pet::getName);

    return m.ptr();
}
```

5.1.3 Compare different approaches

The author has setup an open source repository https://github.com/qingfengxia/python_wrap

> Demonstration, comparison and interopereation of different c++11 class python wrapping methods, like swig, pybind11, binder, cython, cppyy, etc, with CMake integration setup.

Features

- demo and comparison of diff methods
- demonstrate how to build wrap code and link shared object, instead of compile all from sources
- build systems: `cmake`, python `setup.py`
- support C++11 with options setup in cmake and setup.py
- investigate the inter-operation of compiled modules from different methods

5.1.4 Subjective suggestion

For a small project with several header files, pybind11 is recommended to write pyhtonic interface in a manageable way.

For a project with tens of headers, writing configuration file to control binder for code generation is recommended which can automatically generate boilerplate wrapping code, https://github.com/RosettaCommons/binder. https://cppbinder.readthedocs.io/en/latest/basics.html

If the project has no dependency on binary library, but C++ STL. It is recommended to try `cppyy` to generate the interface without writing extra interfacing code. Currently, installation and project integration is a bit difficult.

For large project like https://github.com/LaughlinResearch/pyOCCT, project specific fork of binder, see https://github.com/LaughlinRe
is used with the specific pyOCCT project.

5.2 Mixing C++ and Python in FreeCAD

src/App/FreeCADInit.py : adding mod path into python's `sys.path` and run "Init.Py" in each module. *FreeCADInit.py* is the *Init.py* for FreeCAD python module

Refer to src/Base folder for Interpreter API, like `Base::Interpreter().runCommand()`, `Base::Interpreter().loadModule()`; see src/Base/Interpreter.h, and src/Base/Interpreter.cpp

PythonObject has its own inheritance tree, which is almost parallel with cpp object tree in FreeCAD, see

!DocumentObjectPy___inherit___graph.

see also the source code src/Base/PyObjectBase.h, and src/Base/PyObjectBase.cpp `class BaseExport PyObjectBase : public PyObject`

It is not like other cpp lib that has python wrapper, like VTK another famous 3D visualization. Programmer will use either cpp API or Python API, but not both in one project, usually. The mixture of cpp and python is highly challenging, like when GIL is necessary, reference counting and passing of PyObject. For module developers, pure python developing is a good start point, and anaylising code from other module can also ease the difficulty of hybrid cpp and python programming.

Reminder: FreeCAD is under migration from python2+Qt4 to python3+Qt5. Module developer should

5.2.1 TemplatePyMod as a collection of pure python examples

src/Mod/TemplatePyMod example of pure python module

- src/Mod/TemplatePyMod/DocumentObject.py base class DocumentObject an ViewProvider in python
- src/Mod/TemplatePyMod/FeaturePython.py exampe by making Box part in python
- src/Mod/TemplatePyMod/TaskPanel.py example of making TaskPanel in python,
- src/Mod/TemplatePyMod/Commands.py example of making MenuItem and ToolbarItem in python,

5.2.2 Example of writing Part or Feature in Python

"In FreeCAD we have our own little framework to create Python bindings for cpp classes but these classes are not prepared to be sub-classed in Python."

see example in src/Mod/TemplatePyMod/FeaturePython.py#113

```
def makeBox():
    FreeCAD.newDocument()
    a=FreeCAD.ActiveDocument.addObject("Part::FeaturePython","Box")
    Box(a)
    if FreeCAD.GuiUp:
        ViewProviderBox(a.ViewObject)
```

There must be one cpp DocumentObject derived type like Part::Feature added to Document. Python class must ref/link to the underlying cpp object, during `__init__()` It is the same for `ViewProviderBox(a.ViewObject)`, which has a method of `attach()`. more example can be found in Fem module

5.2.3 What is the limitation of pure python module except for performance?

What is the limitation of pure python module except for performance?

Yorik's response to the question: "If there is no function limitation, pure python could be used to prototype, then coded in cpp.": >"This is exactly what I do now :) I'm starting to convert parts of Draft & Arch modules to cpp. For me the best of the two worlds is hybrid modules such as Path or FEM: A solid, fast base in cpp (python can be very slow for certain types of operations such as processing big lists), and all th eUI tools in python, so they are easy to modify and extend by users (and by programmers too, so you can experiment a lot)"

1) It is possible to develop pure python module without limitation. i.e. do all the work that cpp can do. Pivy is used to generate obj in inventor scene. Performance is a problem, and will threading or GIL will be another constraint?

2) The urpose of module "TemplatePyMod" is basically a repository of examples, src/Mod/TemplatePyMod/DocumentObject.py. There is base class for ViewProvider and DocumentObject for python,

3) SWIG is used only in one place, to generate pivy objects from FreeCADGui. Other code uses C version of or cpp version of pyCXX are used.

4) Which tool can generate DocumentObjectPy.xml, what is the purpose of this XML, it seems export Property to python. https://github.com/FreeCAD/FreeCAD/blob/master/src/App/DocumentObjectPy.xml >The XML files are built by hand, then there is a cmake macro that converts them in .h and .cpp files at build time (an accompanying *PyImp.cpp file must be present)

5.2.4 How python object is serialized

Scripted objects pure python feature One particularity must be understood, those objects are saved in FreeCAD FcStd files with python's JSON module. cPickle is avoid for security reason.

That module turns a python object as a string, allowing it to be added to the saved file. On load, the JSON module uses that string to recreate the original object, provided it has access to the source code that created the object.

5.2.5 DocumentObjectPy

DocumentObjectPy is python export class for App::DocumentObject, "DocumentObjectPy.h" is not manually coded but generated from DocumentObjectPy.xml file, and its implementation is coded in src/App/DocumentObjectPyImp.cpp.

Can `ViewProviderPy, DocumentObjectPy` be subclassed in python?

> Yes, but it is not what FreeCAD usually do. Due to this the normal way is to do things by aggregation (FeaturePythonT<>), if you insist on doing it by sub-classing a further Python wrapper class is needed.

If only new properties are needed for the derived class, just declare FeaturePythonT<> and extend `DocumentObjectPy` in python. see FemSolver example in Fem module analysis.

5.2.6 Python with Qt4/Qt5 support

From the author's point of view, it is recommended to use compatible layer such as "Qt.py" (single python file) or "qtpy" (PyPI package) to support various bindings: "PyQt4, PyQt5, PySide and PySide2". Pyside2 is the preferred binding for LGPL license and it is under heavy development as an official module for Qt.

The usage of "Qt.py" or "qtpy" is similar. `QtWidgets` is a alias to `QtGui` for Qt4.

```
from Qt import QtCore, QtWidgets, QtGui
#from qtpy mport QtCore, QtWidgets, QtGui


from Qt import __binding__
if __binding__ in ('PySide2', 'PyQt5'):
    print('Qt5 binding available')
elif __binding__ in ('PySide', 'PyQt4'):
    print('Qt4 binding available.')
else:
    print('No Qt binding available.')

from QtCore import pyqtSlot  #fine
from QtGui import QIcon
from QtWidgets import QApplication, QWidget, QPushButton, QMessageBox
```

forum discussion is here Would some global shim help the transition to Qt5 and PySide2 for module developers?

Similar to QtPy approached, in FreeCAD, the current solution is **just use PySide even it is actually PySide2**. This compatible layer is generated by repositiory CMakeLists.txt.

```
file(MAKE_DIRECTORY ${CMAKE_BINARY_DIR}/Ext/PySide)
file(WRITE ${CMAKE_BINARY_DIR}/Ext/PySide/__init__.py "# PySide wrapper\n"
"from PySide2 import __version__\n"
"from PySide2 import __version_info__\n")
file(WRITE ${CMAKE_BINARY_DIR}/Ext/PySide/QtCore.py "from PySide2.QtCore import *\n\n"
"#QCoreApplication.CodecForTr=0\n"
"#QCoreApplication.UnicodeUTF8=1\n")
file(WRITE ${CMAKE_BINARY_DIR}/Ext/PySide/QtGui.py  "from PySide2.QtGui import *\n"
"from PySide2.QtWidgets import *\n"
"QHeaderView.setResizeMode = QHeaderView.setSectionResizeMode\n")
file(WRITE ${CMAKE_BINARY_DIR}/Ext/PySide/QtSvg.py  "from PySide2.QtSvg import *\n")
file(WRITE ${CMAKE_BINARY_DIR}/Ext/PySide/QtUiTools.py  "from PySide2.QtUiTools import *\n")
```

5.2.7 Python 3 support is under way

For module developers, it is time to make the module python2 and python3 compatible in 2017.

discussion on python 3 compatibility

Yorik is working on this now, see python 3 fork at github https://github.com/yorikvanhavre/FreeCAD/tree/python3

see discussion on forum: https://forum.freecadweb.org/viewtopic.php?t=12534 according to that discussion, string is the only obstacle but overcomable.

The major changes from python 2 to python 3 excerpt from that discussion

- If using py3, you need minimum 3.3 (otherwise there are some useful unicode<->UTF8 functions not available yet).
- Python3 doesn't have Int and Long anymore. Everything is Long. This is safe to use in py2 too (in python, it is still called int, mind you, but it's a long).
- By far the most important change: All python strings are now unicode. So all strings-related methods are gone. Fortunately since @shoogen and @wmayer already made a great job at spreading the use of UTF8 everywhere, this doesn't give major problems, it even simplifies (no more of that question to use Latin1 or Utf8)
- PyObject->ob_type doesn't exist anymore, use Py_TYPE(PyObject)
- Class definition (PyTypeObject) has changed a lot:
- different way to initialize it (PyVarObject_HEAD_INIT instead of PyObject_HEAD_INIT)
- tp_getattr and tp_setattr slots are being deprecated. Use tp_getattro and tp_setattro instead (they take a PyObject instead of a char*)
- several numeric handlers have been removed: divide (everything is remainder now), long (everything is long now, but the handler to keep is int :?), coerce, oct and hex.

For python code (97% problems in FreeCAD source are print() and key of dict):

- print("something","something else") doesn't work in py2 unless "from **future** import print_function"
- for key, value in dict.iteritems() becomes for key,value in dict.items(), works in py2 too
- except Error,message becomes except Error(message) works in py2 too
- import submodule becomes from . import submodule

5.2.8 Cheat Sheet: Writing Python 2-3 compatible code

A PDF version is here: http://python-future.org/compatible_idioms.pdf

This notebook shows you idioms for writing future-proof code that is compatible with both versions of Python: 2 and 3. It accompanies Ed Schofield's talk at PyCon AU 2014, Writing 2/3 compatible code

Minimum versions: Python 2: 2.6+ Python 3: 3.3+

There is a python module **sixer**, which works as *2to3* but it translates python2 code into python2 and python3 compatible.

5.3 Python wrapping in FreeCAD

5.3.1 Choice of python wrapping in FreeCAD

Although there are several python binding tool recently, when FreeCAD was born, the choice was very limited. The python API of FreeCAD is mostly created by hand. See for example all files ending with *Py.cpp in the source code.

The choice is discussed in forum thread: https://forum.freecadweb.org/viewtopic.php?t=300

> However, the reason not to use SWIG or SIP is that they are a bit overkill and too complex. BTW, from one version to another in SWIG there are always some slight internal changes which makes it very difficult to keep the sources in shape that they work OK with different SWIG versions. We have made this experience with pivy – a Python binding for Coin. This is a constant point of trouble we run into on different platforms Another even more important reason is that we, in most cases, do not want to wrap the interface 1:1 of the Python and C++ class.

- **C and C++ Python API** Both `<python.h>` C API and pyCXX c++API are directly used. Wrapping FreeCAD cpp code is kind of writing C module for python, emphasizing performance.

- **Qt wrapping tool sip** is not a choice, since FreeCAD BaseClass is not derived from QObject. However, it is possible to design all FreeCAD classes derived from QObject with pros and cons. FreeCAD can be run without GUI, so the FreeCAD objects should not depend or are mixed with QObject.
- **swig**, It is used only to generate pivy objects from FreeCADGui. swig code can be found at the end of source file src/Base/Interpreter.cpp There is no stable ABI for wrapping, each time swig upgrade, even a mino upgrade from 3.0 to 3.1, a compilation is needed.
- **boost::python** in FreeCAD 0.17, boost::python is a dependent component for FreeCAD.

https://github.com/llvm-mirror/clang/blob/master/bindings/python/clang/cindex.py)

5.3.2 Direct usage of C API is NOT recommended

Direct usage of C API is NOT recommended, since C API is not compatible for the migration from python 2.7 to python 3.3+

Recently, Python 3.x defined a set of Stable Application Binary Interface(ABI), see https://docs.python.org/3/c-api/stable.html

If module developer wants to mimic some new feature from existent code, understanding of common API in *python.h* is essential

Official document of python C API

- Include Files, Objects, Types and Reference Counts (Introduction)
- The Very High Level Layer (cpp structurs responds to common python objects)
- Reference Counting ()
- Exception Handling (set proper exception handling before return NULL)

general tutorial on tutorialpoint.com, before jumping into FreeCAD source code

5.3.3 PyCXX: supporting Python 2 and Python 3

PyCXX support both python 2 and python 3

It is possible to have common code that can be compiled to work with Python 2 or Python 3.

Use PyCXX V5.5 with PYCXX_PYTHON_2TO3 defined to support Python 2. Use PyCXX V6.0 with PYCXX_PYTHON_2TO3 defined to support Python 3.

The changes from Python 2 to Python 3 that require code changes are:

string is unicode only in Python 3 - Py::String API changed to match python 3 usage byte is for byte data in Python 3 - Py::Bytes added to PyCXX int has been removed - Py::Int has been removed from PyCXX

This means that you will need to:

- Replace Py::Nothing with Py::None - required
- Replace Py::Int with Py::Long - recommended
- Replace Py::LongLong with Py::Long -recommended
- Replace as_std_string() with as_std_string(“encoding”) or as_std_string(NULL) - required
- Replace Py::String that holds non unicode data with Py::Bytes - required
- Because the Py::String and Py::Byte implementations in PyCXX V5.5 allow

5.3.4 Boost.Python and pybind11

Py++ uses GCC C++ compiler to parse C++ source files and allows you to expose C++ code to Python in quick and elegant way using the Boost.Python library.

It uses the following steps to do so: - source code is passed to GCC-XML - GCC-XML passes it to GCC C++ compiler - GCC-XML generates an XML description of a C++ program from GCC’s internal representation. - Py++ uses pygccxml package to read GCC-XML generated file.

Pybind11 is a relative new project, similar with boost.python. Pybind11 uses C++11 new lang features to simply the wrapping process, making it easier to write more pythonic interface. Pybind11 also has a project `binder` to provide automatic wrapping code for Pybind11.

5.3.5 Simplified wrapping by template FeaturePythonT

For module developer who works only at the DocumentObject level, usage of FeaturePythonT could be sufficient without touching `PyObject*`

FeaturePythonT Generic Python feature class which allows to behave every DocumentObject derived class as Python feature – simply by subclassing. FeatureT

5.3.6 Automatically generate wrapping code in FreeCAD

> There are also a lot of helper structures and templates to ease with that tedious process. For example, in the different src/Mod folders, when you see a *Py.xml file together with a* PyImp.cpp, the xml files contains a structure that will automatically generate *Py.h and* Py.cpp files at build time, to which the *.PyImp.cpp will be merged.

It is possible to parse C++ header file to generate the interface definition like the XML file

!DocumentObjectPy___inherit___graph

This file is generated by src/Tools/generateTemaplates/templateClassPyExport.py out of the XML file

Automatic python wrapping code can be generated by python script in building tools.

src/Mod/Part/App/ConePy.xml is are built by hand (which could be generate from text definition file or swig scanning from header file in the future), then there is a cmake macro that converts them in *Py.h and* Py.cpp files at build time (an accompanying *PyImp.cpp file must be present).

In the src/Mod/Part/App/AppPart.cpp, this python type is registered to interpreter `Base::Interpreter().addType(&Part::ConePy::Type,partModule,"Cone");` which is implemented in src/Base/Interpreter.cpp

```
void InterpreterSingleton::addType(PyTypeObject* Type,PyObject* Module, const char * Name)
{
    // NOTE: To finish the initialization of our own type objects we must
    // call PyType_Ready, otherwise we run into a segmentation fault, later on.
    // This function is responsible for adding inherited slots from a type's base class.
    if (PyType_Ready(Type) < 0) return;
    union PyType_Object pyType = {Type};
    PyModule_AddObject(Module, Name, pyType.o);
}
```

Then this cpp type/class is registered into cpp type system in src/Mod/Part/App/AppPart.cpp `Part::Cone::init();`

```
void BaseClass::init(void)
{
    assert(BaseClass::classTypeId == Type::badType() && "don't init() twice!");
    /* Make sure superclass gets initialized before subclass. */
    /*assert(strcmp(#_parentclass_), "inherited"));*/
    /*Type parentType(Type::fromName(#_parentclass_));*/
    /*assert(parentType != Type::badType() && "you forgot init() on parentclass!");*/

    /* Set up entry in the type system. */
    BaseClass::classTypeId =
        Type::createType(Type::badType(),
                         "Base::BaseClass",
                         BaseClass::create);
}
```

5.4 Extending cpp class function in Python

aggregation here means adding function to class without subclassing.

It is possible to extend cpp DocumentObject in Python. see discussion on forum What is relation between Fem/App/FemAnalysis.h and _FemAnalysis.py

5.4.1 Proxy relationship

Proxy is a property of `App::PropertyPythonObject Proxy;`. Both methods defined in Python and cpp will be called, see [] Python needs not to specify which class is derived, just provide the methods(API).

Todo: *This section is not completed!!!* Sequence? derived from *Imp

```
    def attach(self, vobj):
        self.ViewObject = vobj
        self.Object = vobj.Object
        self.bubbles = None
```

The ViewProvider attachment happens here src/Gui/ViewProviderPythonFeature.cpp#L299

```
protected:
    virtual void onChanged(const App::Property* prop) {
        if (prop == &Proxy) {
            if (ViewProviderT::pcObject && !Proxy.getValue().is(Py::_None())) {
                if (!_attached) {
                    _attached = true;
                    imp->attach(ViewProviderT::pcObject);
                    ViewProviderT::attach(ViewProviderT::pcObject);
                    // needed to load the right display mode after they're known now
                    ViewProviderT::DisplayMode.touch();
                    ViewProviderT::setOverrideMode(viewerMode);
                }
                ViewProviderT::updateView();
            }
        }
        else {
            imp->onChanged(prop);
            ViewProviderT::onChanged(prop);
        }
    }
```

5.4.2 `App::FeaturePythonT` in src/App/FeaturePython.h

src/App/FeaturePythonPyImp.h FeaturePyT

```
// Special Feature-Python classes
typedef FeaturePythonT<DocumentObject> FeaturePython;
typedef FeaturePythonT<GeoFeature > GeometryPython;
```

src/App/FeaturePython.h

```
// Helper class to hide implementation details
class AppExport FeaturePythonImp
...
template <class FeatureT>
class FeaturePythonT : public FeatureT
{
...
    /// recalculate the Feature
    virtual DocumentObjectExecReturn *execute(void) {
        try {
            bool handled = imp->execute();
            if (!handled)
                return FeatureT::execute();
        }
        catch (const Base::Exception& e) {
            return new App::DocumentObjectExecReturn(e.what());
        }
        return DocumentObject::StdReturn;
    }
...
private:
    FeaturePythonImp* imp;
    DynamicProperty* props;
    PropertyPythonObject Proxy;
};

// Special Feature-Python classes
typedef FeaturePythonT<DocumentObject> FeaturePython;
typedef FeaturePythonT<GeoFeature    > GeometryPython;
```

src/App/FeaturePython.cpp FeaturePythonImp onChange() execute()

If the Python feature class doesn't have an `execute()` method or if it returns False this method also return false and true otherwise.

```
namespace App {
PROPERTY_SOURCE_TEMPLATE(App::FeaturePython, App::DocumentObject)
template<> const char* App::FeaturePython::getViewProviderName(void) const {
    return "Gui::ViewProviderPythonFeature";
}
template<> PyObject* App::FeaturePython::getPyObject(void) {
    if (PythonObject.is(Py::_None())) {
        // ref counter is set to 1
        PythonObject = Py::Object(new FeaturePythonPyT<DocumentObjectPy>(this),true);
    }
    return Py::new_reference_to(PythonObject);
}
// explicit template instantiation
template class AppExport FeaturePythonT<DocumentObject>;
}
```

why `template <class FeaturePyT> class FeaturePythonPyT : public FeaturePyT` is needed? see src/App/FeaturePythonPyIm
src/App/FeaturePythonPyImp.inl

```
/// Type structure of FeaturePythonPyT
template<class FeaturePyT>
PyTypeObject FeaturePythonPyT<FeaturePyT>::Type = {}
```

5.4.3 Example of aggregation of Fem::FemSolverObject

FemSolverObject is derived from DocumentObject without any property added in C++, but it can be extended in Python.

Look at the template class FeaturePythonT, it is of the form: src/App/FeaturePython.h

```
template <class FeatureT>
class FeaturePythonT : public FeatureT
```

template FeatureT is the parent class of FeaturePythonT.

src/Mod/Fem/App/FemSolverObject.h `class AppFemExport FemSolverObject : public App::DocumentObject` `typedef App::FeaturePythonT<FemSolverObject> FemSolverObjectPython;` FemSolverObjectPython is a type of sub-class of Fem::Fem::FemSolverObject.

src/Mod/Fem/App/FemSolverObject.cpp

```
PyObject *FemSolverObject::getPyObject()
{
    if (PythonObject.is(Py::_None())){
        // ref counter is set to 1
        PythonObject = Py::Object(new DocumentObjectPy(this),true);
    }
    return Py::new_reference_to(PythonObject);
}

// Python feature -----------------------------------------------------

namespace App {
/// @cond DOXERR
PROPERTY_SOURCE_TEMPLATE(Fem::FemSolverObjectPython, Fem::FemSolverObject)
template<> const char* Fem::FemSolverObjectPython::getViewProviderName(void) const {
    return "FemGui::ViewProviderSolverPython";
}

template<> PyObject* Fem::FemSolverObjectPython::getPyObject(void) {
    if (PythonObject.is(Py::_None())) {
        // ref counter is set to 1
        PythonObject = Py::Object(new App::FeaturePythonPyT<App::DocumentObjectPy>(this),true);
    }
    return Py::new_reference_to(PythonObject);
}

// explicit template instantiation
template class AppFemExport FeaturePythonT<Fem::FemSolverObject>;
```

`App::FeaturePythonT<FemSolverObject>` is the explicit template instantiation of FemSolverObject but it is still a cpp type.

Why there are two versions of '`Fem::FemSolverObjectPython::getPyObject(void)`?

NB, in most of cases, PyObject C struct is returned from cpp class methods, but there is one exception `Py::Object` is returned in `Py::Object getObject(void) const;` Py::Object has auto ref counting function, defined in PyCXX.

5.4.4 Gui::ViewProviderPythonFeatureT

src/Gui/ViewProviderPythonFeature.h `Gui::ViewProviderPythonFeatureT< ViewProviderT >` has the same trick with App::PythonFeatureT ViewProviderPythonFeatureImp concrete class is used to hide impl details,which has a private `ViewProviderDocumentObject* object;`

NB, if `imp->setEdit(ModNum)` return true, `ViewProviderT::setEdit(ModNum);` is not called! Why? It is same for unset() and doubleClicked(void)

```
    /// is called by the document when the provider goes in edit mode
    virtual bool setEdit(int ModNum)
    {
        bool ok = imp->setEdit(ModNum);
        if (!ok) ok = ViewProviderT::setEdit(ModNum);
        return ok;
    }
```

5.5 Advanced topics: GIL and manually wrapping

Embedding Python in multi-threaded cpp applications

5.5.1 Example of C API usage

Direct usage of C API is essential for developer writing cpp workbench.

see example in src/Mod/Fem/App/AppFemPy.cpp, and src/Mod/Part/App/AppPartPy.cpp PyCMethodDef is a C structure to define the python methods exported to python users.

`PyObject*` is passed in as argument and returned by C wrapper function. Python type checking and argument validation should be done in this function before try-catch block. In addition, proper exception should be set before `return 0`, which means `*NULL PyObject *`.

An exampel of C wrapper function code is shown here: src/Mod/Part/App/AppPartPy.cpp

```
static PyObject * makeTube(PyObject *self, PyObject *args)
{
    PyObject *pshape;
    double radius;
    double tolerance=0.001;
    char* scont = "C0";
    int maxdegree = 3;
    int maxsegment = 30;

    // Path + radius
    if (!PyArg_ParseTuple(args, "O!d|sii", &(TopoShapePy::Type), &pshape, &radius, &scont, &maxdegree, &maxseg
        return 0;
    std::string str_cont = scont;
    int cont;
    if (str_cont == "C0")
        cont = (int)GeomAbs_C0;
    else if (str_cont == "C1")
        cont = (int)GeomAbs_C1;
    else if (str_cont == "C2")
        cont = (int)GeomAbs_C2;
    else if (str_cont == "C3")
        cont = (int)GeomAbs_C3;
    else if (str_cont == "CN")
        cont = (int)GeomAbs_CN;
    else if (str_cont == "G1")
        cont = (int)GeomAbs_G1;
    else if (str_cont == "G2")
        cont = (int)GeomAbs_G2;
    else
        cont = (int)GeomAbs_C0;

    try {
        const TopoDS_Shape& path_shape = static_cast<TopoShapePy*>(pshape)->getTopoShapePtr()->_Shape;
        TopoShape myShape(path_shape);
        TopoDS_Shape face = myShape.makeTube(radius, tolerance, cont, maxdegree, maxsegment);
        return new TopoShapeFacePy(new TopoShape(face));
    }
    catch (Standard_Failure) {
        Handle_Standard_Failure e = Standard_Failure::Caught();
        PyErr_SetString(PartExceptionOCCError, e->GetMessageString());
        return 0;
    }
}
```

C wrapper functions defined in src/Mod/Part/App/AppPartPy.cpp are registered into an Array of `PyCMethodDef`

```
/* registration table  */
struct PyMethodDef Part_methods[] = {
    {"open"       ,open      ,METH_VARARGS,
     "open(string) -- Create a new document and load the file into the document."},
     ...
```

5.5.2 GIL in [src/App/Interpreter.h]

```
/** If the application starts we release immediately the global interpreter lock
 * (GIL) once the Python interpreter is initialized, i.e. no thread -- including
 * the main thread doesn't hold the GIL. Thus, every thread must instantiate an
 * object of PyGILStateLocker if it needs to access protected areas in Python or
 * areas where the lock is needed. It's best to create the instance on the stack,
 * not on the heap.
 */
class BaseExport PyGILStateLocker
{
public:
    PyGILStateLocker()
    {
        gstate = PyGILState_Ensure();
    }
    ~PyGILStateLocker()
    {
        PyGILState_Release(gstate);
    }

private:
    PyGILState_STATE gstate;
};

/**
 * If a thread holds the global interpreter lock (GIL) but runs a long operation
 * in C where it doesn't need to hold the GIL it can release it temporarily. Or
 * if the thread has to run code in the main thread where Python code may be
 * executed it must release the GIL to avoid a deadlock. In either case the thread
 * must hold the GIL when instantiating an object of PyGILStateRelease.
 * As PyGILStateLocker it's best to create an instance of PyGILStateRelease on the
 * stack.
 */
class BaseExport PyGILStateRelease
{
public:
    PyGILStateRelease()
    {
        // release the global interpreter lock
        state = PyEval_SaveThread();
    }
    ~PyGILStateRelease()
    {
        // grab the global interpreter lock again
        PyEval_RestoreThread(state);
    }

private:
    PyThreadState* state;
};

/** The Interpreter class
 *  This class manage the python interpreter and hold a lot
 *  helper functions for handling python stuff
 */
class BaseExport InterpreterSingleton
{
}
```

Chapter 6

Modular Design of FreeCAD (plugin system)

Modular design is a key design principle for a successful software architecture. Salome platform is an Open source CAE platform with geometry building, meshing and FEM and CFD solver modules. Salome has a dummy plugin module, as a start point of a new module. Samole plugin has a standard folder structure

FreeCAD has the same infrastructure, template, and a python code to generate a new module from the template.

6.1 Quick start

wiki page on how to create a module is definitely the start point! More details can be found:

How to build a module/workbench

6.1.1 Official template for a new module

There are workbench template in official source repository, c++ template folder at `[src/Tools/_TEMPLAT_]` and pure python template `[src/Tools/_TEMPLATPY_]`. Those templates give the module developers the recommended/standard module directory structure and file names.

A new module folder structure with essential code for the new module can be generated by `fcbt` script

6.1.2 `fcbt` to generate a new module

`fcbt` is the acrynom for **FreeCAD build tool**, it can replace dummy names with your module name (`CFD` in my example below) in source names.

usage of fcbt.py And example of output:

```
qingfeng@qingfeng-ubuntu:/opt/FreeCAD/src/Tools$ python fcbt.py

FreeCAD Build Tool
Usage:
   fcbt <command name> [command parameter]
possible commands are:
 - DistSrc         (DS)   Build a source Distr. of the current source tree
 - DistBin         (DB)   Build a binary Distr. of the current source tree
 - DistSetup       (DI)   Build a Setup Distr. of the current source tree
 - DistSetup       (DUI)  Build a User Setup Distr. of the current source tree
 - DistAll         (DA)   Run all three above modules
 - NextBuildNumber (NBN)  Increase the Build Number of this Version
 - CreateModule    (CM)   Insert a new FreeCAD Module in the module directory
```

```
For help on the modules type:
  fcbt <command name> ?

Insert command: CM
Please enter a name for your application:Cfd
Copying files... from _TEMPLATE_ folder and modify them
...
Modifying files...
../Mod/Cfd/InitGui.py
../Mod/Cfd/Init.py
../Mod/Cfd/CMakeLists.txt
../Mod/Cfd/App/PreCompiled.h
../Mod/Cfd/App/AppCfd.cpp
../Mod/Cfd/App/PreCompiled.cpp
../Mod/Cfd/App/CMakeLists.txt
../Mod/Cfd/App/AppCfdPy.cpp
../Mod/Cfd/Cfd.dox
../Mod/Cfd/Gui/PreCompiled.h
../Mod/Cfd/Gui/Workbench.cpp
../Mod/Cfd/Gui/AppCfdGui.cpp
../Mod/Cfd/Gui/PreCompiled.cpp
../Mod/Cfd/Gui/CMakeLists.txt
../Mod/Cfd/Gui/Command.cpp
../Mod/Cfd/Gui/AppCfdGuiPy.cpp
../Mod/Cfd/Gui/Workbench.h
../Mod/Cfd/Gui/Resources/Cfd.qrc
Modifying files done.
Cfd module created successfully.
```

6.1.3 List of essential files in Module folder

- Init.py Module initialization code, will be run during FreeCAD startup

  e.g. add importable and exportable file types, it is optional

- InitGui.py to declare Module's Workbench class

  to insert items into FreeCAD Gui

- Fem.dox Independent Doxygen documentation for this module
- Readme.md Description of this module ,shown directly on github
- CMakeList.txt cmake config file, to define installaton of this module
- App C++ code to generate Fem binary dyanamically linkable lib

  All nonGui code should go here, like classes derived from App::DocumentObject

- Gui C++ code to generate FemGui binary dyanamically linkable lib

  Gui code should go here, like classes derived from TaskView, ViewProvider

C++ code in App subfolder

- App/PreCompiled.h include some headers shared by most source code files
- App/PreCompiled.cpp include some headers shared by most source code files
- App/CMakeLists.txt cmake config file to generate dll or so shared dynamically linkable lib
- Gui/AppFem.cpp init_type,init DocumentObject
- Gui/AppFemPy.cpp register types, methods exported to Python

  ```
  #methods can be accessed in python: import Fem dir(Fem)
  ```

C++ code in Gui subfolder

- Gui/Workbench.h to declare module workbench derived from `Gui::Workbench`

- Gui/Workbench.cpp

- Gui/AppFemGui.cpp

  within function of initFemGui(): - Fem_Import_methods[] - load command.cpp, - workbench and ViewProvider init(), - `Base::Interpreter().loadModule('python modules')` - register preferences pages - load resource, mainly translation

- Gui/AppFemGuiPy.cpp wrapping code to export functions to python

  `/* registration table */ struct PyMethodDef FemGui_Import_methods[]`

- Gui/PreCompiled.h include some headers shared by most source code files
- Gui/PreCompiled.cpp contains single line `#include "PreCompiled.h"`
- Gui/CMakeLists.txt cmake config file to generate dll or so shared dynamically linkable lib
- Gui/Command.cpp to add Toolbar and MenuItem to module workbench
- Gui/Resources/Fem.qrc file contains translattion for Qt widgets
-

The template module is organized similar with other official Module in FreeCAD source in the src/Mod folder. Gui related C++ code is located in "Gui" subfolder, while nonGui code are put into "App" subfolder.

The generate module will be in a minimal runnable/compilable state without any specific functions. Extra source files and code should be included and trimmed by module developer .

Some good example and best practice should be included.

6.2 Workbench framework: key to modular design

6.2.1 Create Workbench step by step

This section (Create Workbench step by step) is a copy of FreeCAD doxygen documentation on workbench

> FreeCAD provides the possibility to have one or more workbenches for a module.
>
> A workbench changes the appearance of the main window in that way that it defines toolbars, items in the toolbox, menus or the context menu and dockable windows that are shown to the user. The idea behind this concept is that the user should see only the functions that are required for the task that he is doing at this moment and not to show dozens of unneeded functions which the user never uses.

Here follows a short description of how your own workbench can be added to a module.

6.2.1.1 Inherit either from Workbench or StdWorkbench

First you have to subclass either `Workbench` or `StdWorkbench` and reimplement the methods `setupMenuBar()`, `setupToolBars()`, `setupCommandBars()` `setupDockWindows()`.

The difference between both classes is that these methods of Workbench are pure virtual while StdWorkbench defines already the standard menus and toolbars, such as the 'File', 'Edit', . . . , 'Help' menus with their common functions.

If your class derives from Workbench then you have to define your menus, toolbars and toolbox items from scratch while deriving from StdWorkbench you have the possibility to add your preferred functions or even remove some unneeded functions.

```
class MyWorkbench : public StdWorkbench
{
 ...
protected:
  MenuItem* setupMenuBar() const
  {
    MenuItem* root = StdWorkbench::setupMenuBar();
    // your changes
    return root;
  }
  ToolBarItem* setupToolBars() const
  {
    ToolBarItem* root = StdWorkbench::setupToolBars();
    // your changes
    return root;
  }
  ToolBarItem* setupCommandBars() const
  {
    ToolBarItem* root = StdWorkbench::setupCommandBars();
    // your changes
    return root;
  }
};
//or

class MyWorkbench : public Workbench
{
 ...
protected:
  MenuItem* setupMenuBar() const
  {
    MenuItem* root = new MenuItem;
    // setup from scratch
    return root;
  }
  ToolBarItem* setupToolBars() const
  {
    ToolBarItem* root = new ToolBarItem;
    // setup from scratch
    return root;
  }
  ToolBarItem* setupCommandBars() const
  {
    ToolBarItem* root = new ToolBarItem;
    // setup from scratch
    return root;
  }
};
```

6.2.1.2 Customizing the workbench

If you want to customize your workbench by adding or removing items you can use the ToolBarItem class for customizing toolbars and the MenuItem class for menus. Both classes behave basically the same. To add a new menu item you can do it as follows

```
MenuItem* setupMenuBar() const
{
  MenuItem* root = StdWorkbench::setupMenuBar();
  // create a sub menu
  MenuItem* mySub = new MenuItem; // note: no parent is given
  mySub->setCommand( "My &Submenu" );
  *mySub << "Std_Undo" << "Std_Redo";
  // My menu
  MenuItem* myMenu = new MenuItem( root );
  myMenu->setCommand( "&My Menu" );
  // fill up the menu with some command items
  *myMenu << mySub << "Separator" << "Std_Cut" << "Std_Copy" << "Std_Paste" << "Separator" << "Std_Undo" << "S
}
```

Toolbars can be customized the same way unless that you shouldn't create subitems (there are no subtoolbars).

6.2.1.3 Register your workbench

Once you have implemented your workbench class you have to register it to make it known to the FreeCAD core system. You must make sure that the step of registration is performed only once. A good place to do it is e.g. in the global function initMODULEGui in AppMODULEGui.cpp where MODULE stands for the name of your module. Just add the line `MODULEGui::MyWorkbench::init();` somewhere there.

6.2.1.4 Create an item for your workbench registry

Though your workbench has been registered now, at this stage you still cannot invoke it yet. Therefore you must create an item in the list of all visible workbenches. To perform this step you must open your InitGui.py (a Python file) and do some adjustments. The file contains already a Python class MODULEWorkbench that implements the Activate() method (it imports the needed library). You can also implement the GetIcon() method to set your own icon for your workbench, if not, the default FreeCAD icon is taken, and finally the most important method GetClassName(). that represents the link between Python and C++. This method must return the name of the associated C++ including namespace. In this case it must the string `ModuleGui::MyWorkbench`. At the end you can change the line from

`Gui.addWorkbench("MODULE design",MODULEWorkbench())` to `Gui.addWorkbench("My workbench",MODULEWorkbench())` or whatever you want.

6.2.1.5 Note

You must make sure to choose a unique name for your workbench (in this example “My workbench”). Since FreeCAD doesn’t provide a mechanism for this you have to care on your own.

6.2.1.6 More details and limitations

One of the key concepts of the workbench framework is to load a module at runtime when the user needs some function that it provides. So, if the user doesn’t need a module it never gets loaded into RAM. This speeds up the startup procedure of FreeCAD and saves memory. At startup FreeCAD scans all module directories and invokes InitGui.py. So an item for a workbench gets created. If the user clicks on such an item the matching module gets loaded, the C++ workbench gets registered and activated.

The user is able to modify a workbench (Edit|Customize). E.g. he can add new toolbars or items for the toolbox and add his preferred functions to them. But he has only full control over “his” toolbars, the default workbench items cannot be modified or even removed.

FreeCAD provides also the possibility to define pure Python workbenches. Such workbenches are temporarily only and are lost after exiting the FreeCAD session. But if you want to keep your Python workbench you can write a macro and attach it with a user defined button or just perform the macro during the next FreeCAD session. Here follows a short example of how to create and embed a workbench in Python

```
w=Workbench()                                         # creates a standard workbench (the same as StdWork
w.MenuText = "My Workbench"                           # the text that will appear in the combo box
dir(w)                                                # lists all available function of the object
FreeCADGui.addWorkbench(w)                            # Creates an item for our workbenmch now
                                                      # Note: We must first add the workbench to run some
                                                      # Then we are ready to customize the workbench
list = ["Std_Test1", "Std_Test2", "Std_Test3"]        # creates a list of new functions
w.appendMenu("Test functions", list)                  # creates a new menu with these functions
w.appendToolbar("Test", list)                         # ... and also a new toolbar
```

6.2.1.7 why StdWorkbench needs to be constructed each time?

```
DockWindowItems* PythonWorkbench::setupDockWindows() const
{
    StdWorkbench wb;
    return wb.setupDockWindows();
}
```

6.2.1.8 why two *workbench* source code?

One in python the other in C++ src/Mod/Fem/Gui/Workbench.cpp src/Mod/Fem/InitGui.py

6.2.2 pure python module is possible like *Plot* module

It is error-prone to mix C++ and Python. Fortunately, it is possible using Python only to develop plugin, *Cfd* or 'plot' workbench is the example.

```
class CfdWorkbench(Workbench):
    "CFD workbench object"
    def __init__(self):
        self.__class__.Icon = FreeCAD.getResourceDir() + "Mod/Fem/Resources/icons/FemWorkbench.svg"
        self.__class__.MenuText = "CFD"
        self.__class__.ToolTip = "CFD workbench"

    def Initialize(self) :
        import Fem
        import FemGui

        import _CommandCfdAnalysis
        import _CommandCfdSolverFoam
        import _CommandCfdSolverControl
        import _CommandCfdResult

        # Post Processing commands are located in FemWorkbench
        cmdlst = ['Cfd_Analysis', 'Fem_ConstraintFluidBoundary', 'Cfd_SolverControl', 'Cfd_Result']
        self.appendToolbar(str(QtCore.QT_TRANSLATE_NOOP("Cfd", "CFD tools")), cmdlst)
        self.appendMenu(str(QtCore.QT_TRANSLATE_NOOP("Cfd", "CFD menu")), cmdlst)

    def GetClassName(self):
        return "Gui::PythonWorkbench"

Gui.addWorkbench(CfdWorkbench())
```

Icon could be XPM embedded into source code, or just pick up one from other module. Python workbench could has its own "Resource" folder under module folder, instead of "Mod/ModName/Gui/Resource".

QT user interface language Translation:

6.2.3 Module Init process

Python *Init.py* registered import and export file types, and "InitGui.py" append command class or other UI elements to module workbench

C++ side registered type and export to python, a similar but much simpler process as [src/App/Application.cpp] and [src/App/ApplicationPy.cpp]

For example, src/Mod/Fem/Gui/AppFemGui.cpp registered all viewProvider types, C++ commands classes defined in command.cpp, load extra python module.

6.3 Module Preference

Each module can have its own preference setting:

6.3.1 Graphical User interface for Preference

`Menu->Preference...` see the details on FreeCAD wiki page: https://www.freecadweb.org/wiki/Preferences_Editor

6.3.2 parameter without Graphical UI

`Tool->Parameter` has some windows register style/parameter setting without design QT dialog ui file [images/FreeCAD_parameter_edit

6.3.3 access preference parameter

The parameter data type and hierarchy can be checked up in `~/.FreeCAD/user.cfg`

For example, unit schema detection is very important in FEA, CFD meshing export and boundary setting `GetParameterGroupByPath() // return Base::Reference<ParameterGrp> instead of ParameterGrp::handle`

```
#include <Base/Parameter.h>
#include <App/Application.h>

/**
 *  Constructs a DlgSettingsUnitsImp which is a child of 'parent', with the
 *  name 'name' and widget flags set to 'f'
 */
DlgSettingsUnitsImp::DlgSettingsUnitsImp(QWidget* parent)
    : PreferencePage( parent ), ui(new Ui_DlgSettingsUnits)
{
    ui->setupUi(this);

    //fillUpListBox();
    ui->tableWidget->setVisible(false);
}


void DlgSettingsUnitsImp::saveSettings()
{
    // must be done as very first because we create a new instance of NavigatorStyle
    // where we set some attributes afterwards
    auto hGrp = App::GetApplication().GetParameterGroupByPath
        ("User parameter:BaseApp/Preferences/Units");
    hGrp->SetInt("UserSchema", ui->comboBox_ViewSystem->currentIndex());
    hGrp->SetInt("Decimals", ui->spinBoxDecimals->value());
    Base::UnitsApi::setDecimals(ui->spinBoxDecimals->value());
}

void DlgSettingsUnitsImp::loadSettings()
{
    auto hGrp = App::GetApplication().GetParameterGroupByPath
        ("User parameter:BaseApp/Preferences/Units");
    ui->comboBox_ViewSystem->setCurrentIndex(hGrp->GetInt("UserSchema",0));
    ui->spinBoxDecimals->setValue(hGrp->GetInt("Decimals",Base::UnitsApi::getDecimals()));
}
```

```
p = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Mod/Draft")
p = p.GetString("TeighaFileConverter")
```

Some example utility functions have been defined in [src/mod/Draft/Draft.py]

```
#getParamType(param) is also defined in this file
def getParam(param,default=None):
    "getParam(parameterName): returns a Draft parameter value from the current config"
    p = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Mod/Draft")
    t = getParamType(param)
    #print("getting param ",param, " of type ",t, " default: ",str(default))
    if t == "int":
        if default == None:
            default = 0
        return p.GetInt(param,default)
    elif t == "string":
        if default == None:
            default = ""
        return p.GetString(param,default)
    elif t == "float":
        if default == None:
            default = 0
        return p.GetFloat(param,default)
    elif t == "bool":
        if default == None:
            default = False
        return p.GetBool(param,default)
    elif t == "unsigned":
        if default == None:
            default = 0
        return p.GetUnsigned(param,default)
    else:
        return None

def setParam(param,value):
    "setParam(parameterName,value): sets a Draft parameter with the given value"
    p = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Mod/Draft")
    t = getParamType(param)
    if t == "int": p.SetInt(param,value)
    elif t == "string": p.SetString(param,value)
    elif t == "float": p.SetFloat(param,value)
    elif t == "bool": p.SetBool(param,value)
    elif t == "unsigned": p.SetUnsigned(param,value)
```

6.3.4 Programming preference in C++ and python

see forum discussion on preference [Interactive preferences page in python?](https://forum.freecadweb.org/viewtopic.php?t=18099>

There exist two ways of adding preferences pages: 1. A static approach where you use the Pref* widgets. The magic happens in the class *PreferenceUiForm* which goes through all found `Pref* widgets` and call their load & save methods.

PrefWidget is the base class for various Qt input widgets, whose source code can be found in src/Gui/PrefWidgets.h

```
class GuiExport PrefWidget : public WindowParameter
{
public:
  void setEntryName( const QByteArray& name );
  QByteArray entryName() const;

  void setParamGrpPath( const QByteArray& path );
  QByteArray paramGrpPath() const;

  virtual void OnChange(Base::Subject<const char*> &rCaller, const char * sReason);
  void onSave();
  void onRestore();

protected:
  /** Restores the preferences
   * Must be reimplemented in any subclasses.
   */
  virtual void restorePreferences() = 0;
  /** Save the preferences
   * Must be reimplemented in any subclasses.
   */
  virtual void savePreferences()    = 0;

  PrefWidget();
  virtual ~PrefWidget();

private:
  QByteArray m_sPrefName;
  QByteArray m_sPrefGrp;

  // friends
  friend class Gui::WidgetFactoryInst;
};
```

example of making preference page UI in python, no extra python code is needed to fill the UI with parameter data or collect/save data into file

2. Sometimes the static approach is not sufficient because you want to implement a special program logic.

```
from PySide import QtGui

class MyPrefPage:
  def __init__(self, parent=None):
    print ("Create pref page")
    self.form = QtGui.QWidget()
    self.form.setWindowTitle("My pref page")
  def saveSettings(self):
    print ("saveSettings")  # FreeCAD.ParamGet('...').setInt(param, value)
  def loadSettings(self):
    print ("loadSettings")  # value = FreeCAD.ParamGet('...').getInt(param)

Gui.addPreferencePage(MyPrefPage,"General")
```

Preference page can be loaded t workbench in Python in InitGui.py [src/mod/Draft/InitGui.py]

```
import Draft_rc
FreeCADGui.addPreferencePage(":/ui/preferences-dxf.ui","Import-Export")
FreeCADGui.addPreferencePage(":/ui/preferences-dwg.ui","Import-Export")
FreeCADGui.addPreferencePage(":/ui/preferences-svg.ui","Import-Export")
FreeCADGui.addPreferencePage(":/ui/preferences-oca.ui","Import-Export")

FreeCADGui.showPreferences("Import-Export",2)
```

6.3.5 User preference Page storage

path for saving user preference is `~/.FreeCAD/user.cfg`, see the xml content below. There is another file called `~/.FreeCAD/system.cfg`, but it is not recommended to edit by module developers

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<FCParameters>

  <FCParamGroup Name="Root">
    <FCParamGroup Name="BaseApp">
      <FCParamGroup Name="Preferences">
        <FCParamGroup Name="Units">
          <FCInt Name="UserSchema" Value="0"/>
          <FCInt Name="Decimals" Value="2"/>
```

to retrieve the setting which is a combobox UI, “0” (standard: mm-kg-s) `FreeCAD.ParamGet("User parameter:BaseApp/Preferences/`

Which is very useful to scale and export geometry, mesh to MKS unit scheme (metre-kg-second).

6.4 Qt specific UI design

6.4.1 FreeCAD Qt designer plugin installation

excerpt from https://www.freecadweb.org/wiki/CompileOnUnix#Qt_designer_plugin

If you want to develop Qt stuff for FreeCAD, you'll need the Qt Designer plugin that provides all custom widgets of FreeCAD. Go to src/Tools/plugins/widget

So far we don't provide a makefile – but calling `qmake plugin.pro` creates it. Once that's done, calling `make` will create the library libFreeCAD_widgets.so. To make this library known to Qt Designer you have to copy the file to `$QTDIR/plugin/designer`

A practical example is found in forum How to save preferences or how to setup Qt Designer `#include "moc_DlgSettingsFemImp.cpp"`

6.4.2 MOC (Qt meta object compiling) ui file compiling

Qt ui file for c++ taskpanel need a compilation, it is automated by CMake [src/Mod/Fem/Gui/CMakeList.txt]

```
set(FemGui_MOC_HDRS

...

TaskFemConstraintForce.h
...

)

fc_wrap_cpp(FemGui_MOC_SRCS ${FemGui_MOC_HDRS})

SOURCE_GROUP("Moc" FILES ${FemGui_MOC_SRCS})
```

python script needs not such a compilation, in-situ parse the ui file by `FreeCADGui.PySideUic.loadUi()`.

```
ui_path = os.path.dirname(__file__) + os.path.sep + "TaskPanelCfdSolverControl.ui"
self.form = FreeCADGui.PySideUic.loadUi(ui_path)
```

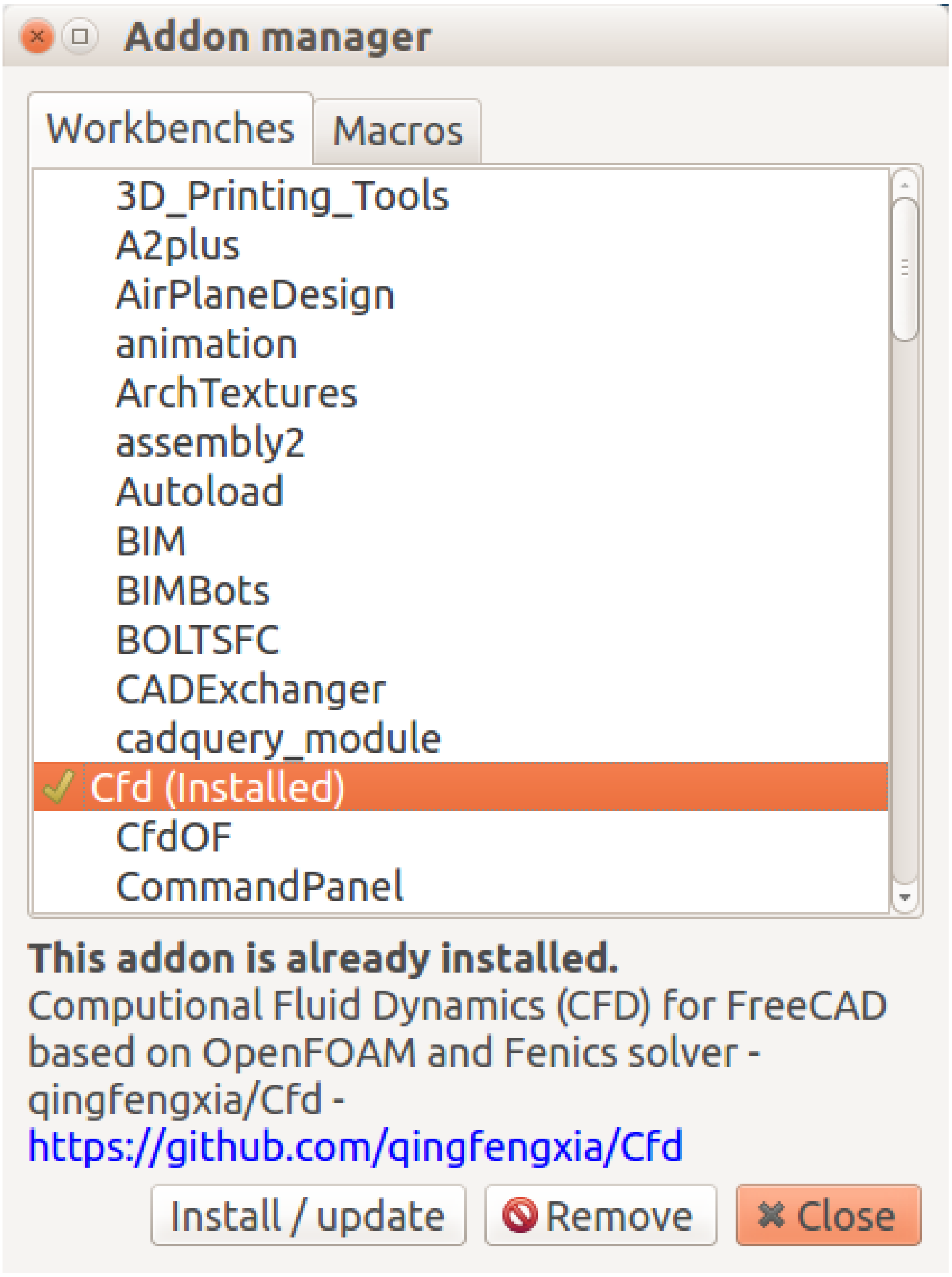
6.5 Install extra module from Addon Manager

Since v0.17 **Addon Manager** is merged into official repo, which makes install and uninstall extra (non-official) module very conveneint. In addition to module management, it can also mange FreeCAD macro files, which are hosted https://github.com/FreeCAD/FreeCAD-macros

Besides modules included in official source code src/Mod, extra modules can be found from **add-ons repository for FreeCAD** https://github.com/FreeCAD/FreeCAD-addons. It is kind of registry of module hosted elsewhere.

Some module extend FreeCAD's traditional CAD functions

- drawing_dimensions: dimensioning for SVG 2D drawing (obsolete, see TechDraw WB)
- nurbs: NURBS curve drawing
- bolts:
- fasteners:
- sheetmetal: metalsheeting
- fcgear: draw gears quickly with parameter



This addon is already installed.
Computional Fluid Dynamics (CFD) for FreeCAD based on OpenFOAM and Fenics solver - qingfengxia/Cfd - https://github.com/qingfengxia/Cfd

Figure 6.1: screenshot for FreeCAD addon manager

- animation: part movement animation

- PluginLoader: browse and install Mod instead of *git+compile*

- parts_library: library for standard components like step motor

- symbols_library:

Some extra modules for CAE:

- Cfd: computional fluid dynamics
- pcb: Printed Circuit Board Workbench
- OpenPLM: as *git for source code* for product design file PLM means product life time management.
- CadQuery:
- CuraEngine: a powerful, fast and robust engine for processing 3D models into 3D printing instruction. For Ultimaker and other GCode based 3D printers. It is part of the larger open source project called “Cura”.

Chapter 7

CAD modules in FreeCAD

7.1 Introduction to FreeCAD's CAD features

Essential modules comparable to commercial tools

- Part: primitive part making and boolean operation
- Import: CAD data exchange ()
- Sketcher: bottom up geometry building from 2D sketch profile (wire) to 3D part
- PartDesign: pattern to design complicate part
- TechDraw: 2D drawing for printout
- Assembly: not quite mature (by default not enabled in build)

CAM + Path: tool path + Cam + Robotics

CAD and CAE infrastructure + Mesh + MeshPart + Point + Surface + Material + JtReader + Measure

more general infrastructure + Plot + Image + spreadsheet

domain specific CAD + Ship: CAD for ship building industry + Arch: architecture design + Idf: data exchange format for architecture design

third-party addons and repo +

7.2 OpenCASCADE and FreeCAD

7.2.1 OpenCasCade Overview

OpenCASCADE Technoloyg (OCCT) is kind of only full-feature, industrial-strength, open source CAD kernel, on which FreeCAD and Salome are built on. OCCT has been released under LGPL in 2013, not from OCC license any more, see http://www.opencascade.com/doc/occt-6.9.0/overview/html/technical_overview.html.

The official (doxygen generated) document is here https://www.opencascade.com/doc/occt-7.3.0/refman/html/index.html

> all OCCT classes organized in hierarchical form corresponding to OCCT structure (module -> toolkit -> package -> class).

7.2.2 Relation between OpenCascade(Salome) and FreeCAD

Table 7.1: Relation between OpenCascade(Salome) and FreeCAD

FreeCAD	OpenCASCADE module	Relation
Base	Module FoundationClasses for math, smart pointer,	no
App	Module ApplicationFramework(OCAF)for Document and object organization	no

FreeCAD	OpenCASCADE module	Relation
Gui	Module Visualization(VIS), Salome platform has the GUI geometry builder module	no
Part	Module ModelingAlgorithms,Module ModelingData	based on
Import	Module DataExchange for brep, step and iges CAD format	based on
Mesh	surface mesh	partially based on
FEM	`Salome mesh (SMESH)` format for volume mesh	partially based on

When FreeCAD was designed in early 2000s, OCAF and VIS is not available, therefore, the FreeCAD team design their own infrastructure modules (Base, App and Gui).

OCCT 's VIS component provides shapes via VTK library, in FreeCAD 3D shape renering is done by Coin3D/openInventor.

7.2.3 TopoDS_Shape

Topology defines relationships between simple geometric entities. A shape, which is a basic topological entity, can be divided into components (sub-shapes):

- Vertex - a zero-dimensional shape corresponding to a point;
- Edge - a shape corresponding to a curve and bounded by a vertex at each extremity;
- Wire - a sequence of edges connected by their vertices;
- Face - a part of a plane (in 2D) or a surface (in 3D) bounded by wires;
- Shell - a collection of faces connected by edges of their wire boundaries;
- Solid - a finite closed part of 3D space bounded by shells;
- Compound solid - a collection of solids connected by faces of their shell boundaries.

7.2.4 Salome platform

Salome 2019 has builder module, comparable with FreeCAD.

7.3 Part Module

`Part` module is coded in C++ for better performance (another reason: there is no python wrapping to the underlying CAD kernel: OpenCASCASE when FreeCAD was designed), but there is the official example of pure python implemented Feature like `Plot`.

7.3.1 Important headers in Part Module

7.3.2 src/Mod/Part/App/PartFeature.h

```
class PartExport Feature : public App::GeoFeature
class FilletBase : public Part::Feature
class PartExport FeatureExt : public Feature
std::vector<Part::cutFaces> Part::findAllFacesCutBy(
        const TopoDS_Shape& shape, const TopoDS_Shape& face, const gp_Dir& dir)
PartExport
const bool checkIntersection(const TopoDS_Shape& first, const TopoDS_Shape& second,
                             const bool quick, const bool touch_is_intersection);

}
```

src/Mod/Part/App/PartFeature.cpp

```
Feature::Feature(void)
{
    ADD_PROPERTY(Shape, (TopoDS_Shape()));
}

PyObject *Feature::getPyObject(void)
{
    if (PythonObject.is(Py::_None())){
        // ref counter is set to 1
        PythonObject = Py::Object(new PartFeaturePy(this),true);
    }
    return Py::new_reference_to(PythonObject);
}

TopLoc_Location Feature::getLocation() const
{
    Base::Placement pl = this->Placement.getValue();
    Base::Rotation rot(pl.getRotation());
    Base::Vector3d axis;
    double angle;
    rot.getValue(axis, angle);
    gp_Trsf trf;
    trf.SetRotation(gp_Ax1(gp_Pnt(), gp_Dir(axis.x, axis.y, axis.z)), angle);
    trf.SetTranslationPart(gp_Vec(pl.getPosition().x,pl.getPosition().y,pl.getPosition().z));
    return TopLoc_Location(trf);
}
```

src/Mod/Part/App/PartFeature.cpp

```
/** 2D Shape
  * This is a specialized version of the PartShape for use with
  * flat (2D) geometry. The Z direction has always to be 0.
  * The position and orientation of the Plane this 2D geometry is
  * referenced is defined by the Placement property. It also
  * has a link to a supporting Face which defines the position
  * in space where it is located. If the support is changed the
  * static method positionBySupport() is used to calculate a
  * new position for the Part2DObject.
  * This object can be used stand alone or for constraint
  * geometry as its descend Sketcher::SketchObject .
  */

class PartExport Part2DObject : public Part::Feature
```

7.3.3 Sketcher Module: 2D Geometry

parameterization and constraint is constr

src/Mod/Sketcher/App/Sketch.h collection of Part::Geometry::Segment and constraint, Base::Persistence src/Mod/Sketcher/App/Sketcl
Part::Part2DObject (derived from App::DocumentObject) Its own PlaneGCS algorithm

7.3.4 PartDesign Module: advanced 3D model building

src/Mod/PartDesign/App/Feature.h

```
class PartDesignExport Feature : public Part::Feature static TopoDS_Shape getSolid(const TopoDS_Shape&);
```

src/Mod/PartDesign/App/FeaturePad.h FeaturePad<- FeatureAdditive <- SketchBased <- PartDesign::Feature

```
App::PropertyLinkSub UpToFace; // refer to face (subfeature) of another Feature
App::PropertyLink Sketch; // 2D sketch for Pad
```

src/Mod/PartDesign/App/FeatureDressUp.h App::PropertyLinkSub Base; // class PartDesignExport Face : public Part::Part2DObject

7.3.5 PartFeature module

7.3.6 TechDraw

7.4 Mesh and Point modules

Chapter 8

FEM Module Source Code Analysis

Acknowledge of Fem module developers: Bernd, PrzemoF, etc., of course, three core developers.

8.1 Introduction of Fem Module

This module is usable in v0.16 (install netgen and calculix first) and extended dramatically to multiple CAE solver support in v0.17.

Basically, the whole process is to mesh the solid part into small element (currently tetragen only), add boundary condition and material information, write all these information into case file that external solver can accept, launch the external solver (only Calculix is supported for the time being), finally load result from solver output file and show result in FreeCAD workbench.

Official wiki on Fem installation

Official wiki on Fem module

Official wiki on Fem tutorial

For example, error message "CalculiX ccx binary not found! Please set it manually in FEM preferences." is solved by

For Ubuntu 64bit linux, ccx_2.10 single file executable can be downloaded from http://www.dhondt.de/ the dependency `libgfortran.so` can be made available from symbolic link:

```
ln -s <source_path> /lib/x86_64-linux-gnu/libgfortran.so
```

8.2 How is Fem module designed

Although most of the class started with the prefix of "Fem", they works for various CAE application(mechanical, fluidic, electromagnetic).

- FemAnalysis: DocumentObjectGroup derived container hosting the FEM specific DocumentObjects
- Part: geometry to be solved
- FemMaterial: physical properties for the material to be solved
- FemConstrait: variant derived class to apply node, edge, face, volume constraint for CAE problem
- FemMesh: meshing is based on Salome Mesh (SMesh) library, currently netgen only, which is general meshing tool for Fem and CFD.
- FemSolver: Calculix Fem solver is the first usable solver in 0.16, while z88 is in shape in 0.17
- FemResult: ViewProviderFemMesh can color stress in FemMesh, displacement is represented in moved FemMesh
- VTK Post processing: VTK pipeline is implemented in 0.17

In CMakeList.txt, Netgen is not activated by default. It should be activated by cmake/cmake-gui.

8.2.1 FemAppPy.cpp: mesh and result file open and export

Fem module specific file types import and export, this is implemented in [src/Mod/Fem/App/FemAppPy.cpp] Various mesh format can be imported and exported, see the list in FreeCAD wiki for Fem

```
class Module : public Py::ExtensionModule<Module>
{
public:
    Module() : Py::ExtensionModule<Module>("Fem")
    {
        add_varargs_method("open",&Module::open,
            "open(string) -- Create a new document and a Mesh::Import feature to load the file into the docume
        );
        add_varargs_method("insert",&Module::insert,
            "insert(string|mesh,[string]) -- Load or insert a mesh into the given or active document."
        );
        add_varargs_method("export",&Module::exporter,
            "export(list,string) -- Export a list of objects into a single file."
        );
        add_varargs_method("read",&Module::read,
            "Read a mesh from a file and returns a Mesh object."
        );
        add_varargs_method("show",&Module::show,
            "show(shape) -- Add the shape to the active document or create one if no document exists."
        );
        initialize("This module is the Fem module."); // register with Python
    }
```

8.2.2 AppFemGui.cpp `initFemGui()`

functionality of src/Mod/Fem/Gui/AppFemGui.cpp

- `FemGui_Import_methods[]` all method in python module FemGui should be registered
- load commands defined in command.cpp,
- workbench and ViewProvider types init(),
- `Base::Interpreter().loadModule('some python modules in Mod/Fem folder')`
- Register preferences pages `new Gui::PrefPageProducer<FemGui::DlgSettingsFemImp> ("FEM");`
- load resource, mainly icons and translation

8.2.3 Communication of App Object and Gui Object

src/Mod/Fem/Gui/TaskDriver.h `class TaskDriver : public Gui::TaskView::TaskBox` src/Mod/Fem/Gui/TaskDriver.cpp

8.2.4 Selection, SelectionGate and SelectionFilter

todo: This necessity of SelectionGate.py should be explained in GUI chapter

8.2.5 When python scripts should be loaded?

In short, python scripts for Fem should be loaded/imported in `InitGui.py` to avoid cyclic dependency.

see Forum discussion: cyclic dependency FemCommands and FemGui modules

There seems a cyclic dependency. When you try to load FemCommands.py it internally tries to load FemGui. However, at this stage FemCommands.py is not yet loaded and FemGui also tries to load FemCommands.py.

Then there are two flaws inside initFemGui:

1. Base::Interpreter().loadModule() MUST be inside a try/catch block and in case an exception occurs the initFemGui must be aborted and an import error must be raised. Fixed with git commit abd6e8c438c

2. The FemGui must not be initialized before dependent modules are fully loaded. Fixed with git commit 60c8180079f20

The latter fix currently causes the FemGui not to load any more but that's because of the cyclic dependency. IMO, there are two ways to fix:

1. Do not load any of the Python module inside initFemGui because I don't see why they should be needed there. It's much better to move this to the Initialize() method of the Workbench class (InitGui.py)
2. Alternatively, make sure that the Python modules loaded inside initFemGui does not load FemGui in the global scope but do it locally where it's really needed.

8.3 Key classes analysis

In the previous chapter, we have discussed workbench related classes and source files, like: [src/Mod/Fem/App/FemApp.cpp], [src/Mod/Fem/Gui/Workbench.h], [src/Mod/Fem/Gui/Commands.cpp] Here, the FEM specific object are analysed. Key classes except FemMesh and FemConstraint are explained.

8.3.1 src/Mod/Fem/App/FemAnalysis.h DocumentObjectGroup

This is the container DocumentObject in this module. It is NOT a DocumentGroup object before the commit (id ce68094, on Oct 2017 by wwmayer, `inherit FemAnalysis from DocumentObjectGroup and remove Member property`, see https://github.com/FreeCAD/FreeCAD/commit/ce6809415b45d760961eddb58e7c4544b602495a). After refactering, FemAnalysis is derived from `DocumentObjectGroup` directly, the previous `Member` attribute is replaced by `Group`.

New FemObjects will be inserted by `FemGui.GetActiveAnalhysis().addObject(...)`, instead of python list operation on `Member`.

It accepts drag-and-drop of only FEM specific DocumentObjects. Only Fem related DocumentObject can be dragged into, see *ViewProviderFemAnalysis::canDragObject* in src/Mod/Fem/Gui/ViewProviderAnalysis.h. And any new Fem specific DocumentObject should registered here.

```
bool ViewProviderFemAnalysis::canDragObject(App::DocumentObject* obj) const
{
    if (!obj)
        return false;
    if (obj->getTypeId().isDerivedFrom(Fem::FemMeshObject::getClassTypeId()))
        return true;
    else if (obj->getTypeId().isDerivedFrom(Fem::FemSolverObject::getClassTypeId()))
        return true;
    else if (obj->getTypeId().isDerivedFrom(Fem::Constraint::getClassTypeId()))
        return true;
    else if (obj->getTypeId().isDerivedFrom(Fem::FemSetObject::getClassTypeId()))
        return true;
    else if (obj->getTypeId().isDerivedFrom(Base::Type::fromName("Fem::FeaturePython")))
        return true;
    else if (obj->getTypeId().isDerivedFrom(App::MaterialObject::getClassTypeId()))
        return true;
    else
        return false;
}
```

It has no 3D representation in Inventor/Coin scenegraph, different from FemMeshObject or Fem::Constraint. It has an Documdent Observer in GUI part. see src/Mod/Fem/Gui/ActiveAnalysisObserver.h It is a singleton instance `static ActiveAnalysisObserver* instance();`, from which FemGui.getActiveAnalysis() is possible from python.

see `void ActiveAnalysisObserver::setActiveObject(Fem::FemAnalysis* fem)` in src/Mod/Fem/Gui/ActiveAnalysisObserver.cp for activeView and activeDocument are managed

```
namespace FemGui {
```

```
class ActiveAnalysisObserver : public App::DocumentObserver
{
public:
    static ActiveAnalysisObserver* instance();

    void setActiveObject(Fem::FemAnalysis*);
    Fem::FemAnalysis* getActiveObject() const;
```

8.3.2 [src/Mod/Fem/FemMaterial.py]

“MechanicalMaterial” is refactored into a general “FemMaterial”

Command and TaskPanel classes are implemented in FemWorkBench in Python [src/Mod/Fem/FemMaterial.py] material definition data file *.FCMat (not XML but ini style, imported by ConfigParser) is located in the Material module src/Mod/Material/StandardMaterial FluidMaterial is not defined. Water and air should be defined in another folder, src/Mod/Material/FluidMaterial Both these two type of materials should contain more properties, to support other CAE solver, like Eletromagnetic simulation.

```
def makeMechanicalMaterial(name):
    '''makeMaterial(name): makes an Material
    name thereecfore is a material name or an file name for a FCMat file'''
    obj = FreeCAD.ActiveDocument.addObject("App::MaterialObjectPython", name)
    _MechanicalMaterial(obj)
    if FreeCAD.GuiUp:
        _ViewProviderMechanicalMaterial(obj.ViewObject)
    return obj
```

8.3.3 FemResultObject: a good example to create new object

[src/Mod/Fem/Gui/FemResultObject.h] defined several properties to hold data like: Time, Temperature, Displacement,etc. Result mesh can be different from mesh written to solver. It is defined only for solid mechanics, not for fluid dynamics.

This is class has implemented FeatureT<> template, thereby, it can be extended in python into CfdResult for CFD module.

This class defines necessary property to show result, e.g. contour, in 3D scene. This class should be extended in python to deal with result from different solver (different result file type).

Idealy, this class should be a container without any domain specific properties, just like FemSolverObject, mechanical field properties should be made in Python

Bottom-up analysis of this Object:

1. FemResultObject is derived from DocumdentObject with some properties, defined in src/Mod/Fem/App/FemResultObject.h

A list of property can be shared by CFD or Mechancial analysis implemented in src/Mod/Fem/App/FemResultObject.cpp. Most of code is standard, with the defined properties instantiated in constructor.

```
    ADD_PROPERTY_TYPE(Mesh,(0), "General",Prop_None,"Link to the corresponding mesh");
    ADD_PROPERTY_TYPE(NodeNumbers,(0), "Data",Prop_None,"Numbers of the result nodes");
    ADD_PROPERTY_TYPE(Stats,(0), "Fem",Prop_None,"Statistics of the results");
    ADD_PROPERTY_TYPE(Time,(0), "Fem",Prop_None,"Time of analysis incement");
```

Mechancial analysis should be moved into [src/Mod/Fem/FemResult.py]

```
    App::PropertyIntegerList ElementNumbers;
    /// Link to the corresponding  mesh
    App::PropertyLink Mesh;
    /// Stats of analysis
    App::PropertyFloatList Stats;
    /// Displacement vectors of analysis
    App::PropertyVectorList DisplacementVectors;
    /// Lengths of displacement vestors of analysis
    App::PropertyFloatList DisplacementLengths;
    /// Von Mises Stress values of analysis
    App::PropertyFloatList StressValues;
```

2. ViewProvider: [src/Mod/Fem/Gui/ViewProviderResult.h] and [src/Mod/Fem/Gui/ViewProviderResult.cpp]

3. add Command class and appended to workbench menu in Python

[src/Mod/Fem/_CommandShowResult.py]

```
class _CommandMechanicalShowResult:
    "the Fem JobControl command definition"
    def GetResources(self):
        return {'Pixmap': 'fem-result',
                'MenuText': QtCore.QT_TRANSLATE_NOOP("Fem_Result", "Show result"),
                'Accel': "S, R",
                'ToolTip': QtCore.QT_TRANSLATE_NOOP("Fem_Result", "Show result information of an analysis")}

    def Activated(self):
        self.result_object = get_results_object(FreeCADGui.Selection.getSelection())

        if not self.result_object:
            QtGui.QMessageBox.critical(None, "Missing prerequisite", "No result found in active Analysis")
            return

        taskd = _ResultControlTaskPanel()
        FreeCADGui.Control.showDialog(taskd)

    def IsActive(self):
        return FreeCADGui.ActiveDocument is not None and results_present()
```

In this class, `sPixmap = "fem-result"` and helper function *get_results_object* is worth of explanation src/Mod/Fem/Gui/Resources SVG file naming: lowercase with module name as suffix, connected with dash

```
def get_results_object(sel):
    if (len(sel) == 1):
        if sel[0].isDerivedFrom("Fem::FemResultObject"):
            return sel[0]

    for i in FemGui.getActiveAnalysis().Member:
        if(i.isDerivedFrom("Fem::FemResultObject")):
            return i
    return None
```

4. TaskView: defined in python: [src/Mod/Fem/_TaskPanelShowResult.py]

5. python script to read result data file: [src/Mod/Fem/ccxFrdReader.py] and other solver result writer

8.3.4 FemResult to VTK pipeline

src/Mod/Fem/App/PropertyPostDataObject.cpp src/Mod/Fem/App/FemPostObject.cpp

loading FemResult into pipe happens in Command Class `CmdFemPostPipelineFromResult`

```
//poly data is the only data we can visualize, hence every post processing object needs to expose it
class AppFemExport FemPostObject : public App::GeoFeature
{

        Fem::PropertyPostDataObject Data;
}

class AppFemExport FemPostFilter : public Fem::FemPostObject
{

        App::PropertyLink Input;
}

class AppFemExport FemPostPipeline : public Fem::FemPostFilter
{
    App::PropertyLinkList       Filter;
    App::PropertyLink           Functions;
    App::PropertyEnumeration    Mode;
}


class AppFemExport FemPostFunction : public App::DocumentObject
```

8.4 FemConstraint

- DocumentObject: FemConstraint is derived from
- ViewProvider: ViewProviderFemConstraint
- TaskPanel and TaskDlg:
- CMakeList.txt: adding those source files into CMakeList.txt
- SVG icon file in researce folder and XML file

Actually drawing is defined in ViewProviderFemConstraint.cpp

```
createSymmetry(sep, HEIGHT, WIDTH);
createPlacement(pShapeSep, b, SbRotation(SbVec3f(0,1,0), ax));
// gear , change colorProperty, it is a new , aspect ratio, it is new constraint
```

8.4.1 FemConstraint: base class for all Constraint Type

FemConstraint is derived from DocumentObject

8.4.2 ViewProviderFemConstraint: drawing in inventor scene

This is an DocumentObject with basic 3D Inventor, and a good start point to learn drawing in 3D scence. Fem::Constraint is the base class for all the other specific constraints, or boundary conditions, in other domain like CFD.

```
class FemGuiExport ViewProviderFemConstraint : public Gui::ViewProviderGeometryObject
```

```
#define CUBE_CHILDREN 1

void ViewProviderFemConstraint::createCube(SoSeparator* sep, const double width, const double length, const do
{
    SoCube* cube = new SoCube();
    cube->width.setValue(width);
    cube->depth.setValue(length);
    cube->height.setValue(height);
    sep->addChild(cube);
}

SoSeparator* ViewProviderFemConstraint::createCube(const double width, const double length, const double heigh
{
    SoSeparator* sep = new SoSeparator();
    createCube(sep, width, length, height);
    return sep;
}

void ViewProviderFemConstraint::updateCube(const SoNode* node, const int idx, const double width, const double
{
    const SoSeparator* sep = static_cast<const SoSeparator*>(node);
    SoCube* cube = static_cast<SoCube*>(sep->getChild(idx));
    cube->width.setValue(width);
    cube->depth.setValue(length);
    cube->height.setValue(height);
}
```

src/Mod/Fem/Gui/ViewProviderFemConstraintPressure.h draws symbol to represent constrain attachment in 3D view scene

8.4.3 TaskFemConstraint

src/Mod/Fem/Gui/TaskFemConstraint.h `onChange()`: Constraint only record geometry reference, not femCellSet, `accept()` add into Analysis src/Mod/Fem/Gui/TaskFemConstraintPressure.cpp

`#include "moc_TaskFemConstraintPressure.cpp"`

[/src/Mod/Fem/Gui/TaskFemConstraintPressure.h] task panel to select geometry face the pressure constrain is applied to, also the pressure magnitude and direction.

`class TaskFemConstraintPressure : public TaskFemConstraint`

`class TaskDlgFemConstraintPressure : public TaskDlgFemConstraint` accept/reject/open

8.5 Import and export file formats

src/Mod/Fem/App/FemMesh.cpp `write()` and `read()` support export and import various meshing formats, see wiki https://www.freecadweb.org/wiki/index.php?title=FEM_Mesh The meshing IO python API, as used by the FreeCAD GUI, is implemented in src/Mod/Fem/App/AppFemPy.cpp and compiled into `open()` and `export` Fem.so python module.

The import and export file types are registered in [src/Mod/Fem/Init.py]. Various mesh types are auto detected by file suffix, and corresponding mesh import and export c++ API are called accordingly. For example, VTK meshing and result import and export are implemented by the author, thus the following code is added into [src/Mod/Fem/Init.py].

```
if("BUILD_FEM_VTK" in FreeCAD.__cmake__):
    FreeCAD.addImportType("FEM CFD Unstructure Mesh (*.vtk *.vtu)", "Fem")
    FreeCAD.addExportType("FEM CFD Unstructure Mesh (*.vtk *.vtu)", "Fem")
    FreeCAD.addImportType("FEM results (*.vtk *.vtu)", "importVTKResults")
    FreeCAD.addExportType("FEM CFD Result in VTK format (*.vtk *.vtu)", "importVTKResults")
```

The API `FreeCAD.addImportType` takes two parameters, the first is the text hint/filter used by FileDialog during import and export, the second is the python module defined the file import and export functions such as `open(), export(), insert()`. Since the VTK mesh or vtk result can not be distinguished from file suffix, another new python module [src/Mod/Fem/importVTKResults.py] is needed.

This module can serve as a template for new file types support. NB,the file file name `importVTKResults.py` also contains the export API.

8.6 FemMesh, based on Salome SMESH

SMESH: Salome Mesh, supporting both FEM and CFD meshing. Python script is possible in Salome platform.

8.6.1 working with external meshing tools

It is possible to set up boundary condition with imported mesh, given the original part is still in the document file. Unfortunately, boundary facets are not importable into FreeCAD (new data structure is needed to hold such data within mesh), thereby boundary face mesh is extracted by distance detection of meshcell with 3D part surfaces in FreeCAD.

Procedure: 1. export geometry to external meshing tool like Ansys and Salome for advanced function like boundary layer setup, which is not available in FreeCAD yet. 2. mesing and export into a file format supported by FreeCAD import (vtk, unv) or supported by CAE solver directly for case setup. 3. imported back to FreeCAD for FEM/CFD case setup within FreeCAD, then following the FEM module working procedure

8.6.2 src/Mod/Fem/App/FemMesh.h

FreeCAD Fem mesh is based on 3party lib: SMESH, the meshing facility used in Salome. This SMESH is powerful but also chanllenging. It is a deep water zone, just ignore this class if you are not going to extend Fem meshing facility.

```
SMDS_Mesh
SMESH_Mesh
SMESHDS_Mesh
SMESH_SMDS.hxx
```

Important classes:

```
class AppFemExport FemMesh : public Data::ComplexGeoData
class AppFemExport FemMeshObject : public App::GeoFeature
class AppFemExport FemMeshShapeObject : public FemMeshObject
class AppFemExport FemMeshShapeNetgenObject : public FemMeshShapeObject  //with Fineness property

class AppFemExport PropertyFemMesh : public App::PropertyComplexGeoData
class HypothesisPy : public Py::PythonExtension<HypothesisPy>
```

8.6.3 [src/Mod/Fem/FemMeshTools.py]

8.6.4 Mesh generation by Tetgen in C++

Mesh is generated and updated in src/Mod/Fem/Gui/TaskTetParameter.h src/Mod/Fem/Gui/TaskDlgMeshShapeNetgen.cpp

`App::DocumentObjectExecReturn *FemMeshShapeNetgenObject::execute(void)` it does not call super class method FemMeshShapeObject::compute(), defined in src/Mod/Fem/App/FemMeshShapeObject.cpp which is surface squash mesh

Example of PropertyEnumeration: Fineness

```
    ADD_PROPERTY_TYPE(Fineness,(2),         "MeshParams",Prop_None,"Fineness level of the mesh");
    Fineness.setEnums(FinenessEnums);
const char* FinenessEnums[]= {"VeryCoarse","Coarse","Moderate","Fine","VeryFine","UserDefined",NULL};
```

src/Mod/Fem/App/FemMeshShapeNetgenObject.cpp

FemMeshShapeNetgenObject.cpp has no python corresponding object, to set and recompute mesh in python??? src/Mod/Fem/Gui/TaskDlgMeshShapeNetgen.cpp accept() should have some macro recording code like TaskFemConstraintForce's src/Mod/Fem/Gui/TaskTetParameter.h

8.6.5 Mesh generation by Gmsh in Python

With Gmsh, it is possible to specify mesh length scale for subfeatures like faces/edges. FemMeshRegion.py and FemMeshGroup.py

[src/Mod/Fem/FemMeshGmsh.py]

```
def makeFemMeshGmsh(name="FEMMeshGMSH"):
    '''makeFemMeshGmsh(name): makes a GMSH FEM mesh object'''
    obj = FreeCAD.ActiveDocument.addObject("Fem::FemMeshObjectPython", name)
    _FemMeshGmsh._FemMeshGmsh(obj)
    if FreeCAD.GuiUp:
        import _ViewProviderFemMeshGmsh
        _ViewProviderFemMeshGmsh._ViewProviderFemMeshGmsh(obj.ViewObject)
    return obj
```

Gmsh is supported mainly by [src/Mod/Fem/_FemMeshGmsh.py] which defines various properties for gmsh settigns.

[src/Mod/Fem/FemMeshToolGmsh.py] will find the executable binary `gmsh` and write input file `*.geo`, which contains all info needed to generate a mesh

```
def create_mesh(self):
    print("\nWe gone start GMSH FEM mesh run!")
    print(' Part to mesh: Name --> ' + self.part_obj.Name + ', Label --> ' + self.part_obj.Label + ', ShapeTyp
    print(' CharacteristicLengthMax: ' + str(self.clmax))
    print(' CharacteristicLengthMin: ' + str(self.clmin))
    print(' ElementOrder: ' + self.order)
    self.get_dimension()
    self.get_tmp_file_paths()
    self.get_gmsh_command()
    self.get_group_data()
    self.write_part_file() #self.part_obj.Shape.exportBrep(self.temp_file_geometry)
    self.write_geo()  # gmsh input file, containing all info needed  for gen mesh
    error = self.run_gmsh_with_geo()  # gmsh  *.geo
    self.read_and_set_new_mesh()  # load unv file into FemMeshObject
    return error
```

8.6.6 FemSetObject: base class to group submesh

src/Mod/Fem/Gui/TaskCreateNodeSet.cpp nodeset

```
FemSetObject::FemSetObject()
{
    ADD_PROPERTY_TYPE(FemMesh,(0), "MeshSet link",Prop_None,"MeshSet the set belongs to");
}
```

8.6.6.1 FemNodeSet as group of element for constraint

src/Mod/Fem/Gui/TaskCreateNodeSet.cpp nodeset

8.7 FemResult and VTK based post-processing pipeline

Notably, fem result can be view in `showResult` taskpanel, and also it can be exported as vtk file and viewed in paraview.

8.7.1 VTK Pipeline

related files:

src/Mod/Fem/App/FemPostObject.h src/Mod/Fem/App/FemPostPipeline.h src/Mod/Fem/App/FemPostFilter.h src/Mod/Fem/App/FemPostFunction.h

Task panel and view providers in src/Mod/Fem/Gui

It could be thought of miniature paraview pipeline. Implemented in cpp only, perhaps for speed concern.

8.8 PreferencePage

8.8.1 Create preference page in C++

1. implemented related cpp and ui files:

- src/Mod/Fem/Gui/DlgSettingsFemGmsh.ui

- src/Mod/Fem/Gui/DlgSettingsFemGmshImp.h
- src/Mod/Fem/Gui/DlgSettingsFemGmshImp.cpp

```
#include "PreCompiled.h"

#include "Gui/Application.h"
#include "DlgSettingsFemGmshImp.h"
#include <Gui/PrefWidgets.h>

using namespace FemGui;

DlgSettingsFemGmshImp::DlgSettingsFemGmshImp( QWidget* parent )
  : PreferencePage( parent )
{
    this->setupUi(this);
}

DlgSettingsFemGmshImp::~DlgSettingsFemGmshImp()
{
    // no need to delete child widgets, Qt does it all for us
}

void DlgSettingsFemGmshImp::saveSettings()
{
    cb_gmsh_binary_std->onSave();
    fc_gmsh_binary_path->onSave();
}

void DlgSettingsFemGmshImp::loadSettings()
{
    cb_gmsh_binary_std->onRestore();
    fc_gmsh_binary_path->onRestore();
}

/**
 * Sets the strings of the subwidgets using the current language.
 */
void DlgSettingsFemGmshImp::changeEvent(QEvent *e)
{
    if (e->type() == QEvent::LanguageChange) {
    }
    else {
        QWidget::changeEvent(e);
    }
}

#include "moc_DlgSettingsFemGmshImp.cpp"
```

The implementation is surprisingly convenient, just calling onSave() and onRestore() methods of standard PrefWidget defined in [src/Gui/PrefWidgets.h]. This UI file uses some FreeCAD costumed widgets, e.g. `<widget class="Gui::PrefCheckBox" name="cb_int_editor">` Those PrefWidgets needs to be registered into QtDesigner. In short, You need to compile [src/Tools/plugins/widget] and register that library with Qt-designer in order to get the FreeCAD-specific widgets in Qt-designer."

2. register page in [Mod/Fem/Gui/AppFemGui.cpp]

```
// register preferences pages

 new Gui::PrefPageProducer<FemGui::DlgSettingsFemGeneralImp> (QT_TRANSLATE_NOOP("QObject","FEM"));
```

3. Icons of the Preference

“preferences-fem.svg” must be in icon search path, defined in [Mod/Fem/Gui/AppFemGui.cpp]

8.8.2 Create preference page in Python

src/Gui/PrefWidgets.h preference widgets are not wrapped into python, therefore, can not been used by Python

1. create a general QDialog class and implement `saveSettings(), loadSettings()`

```
class CfdPreferencePage:
    def __init__(self):
        ui_path = os.path.join(os.path.dirname(__file__), "CfdPreferencePage.ui")
        self.form = FreeCADGui.PySideUic.loadUi(ui_path)
```

2. register this preference page in `InitGui.py`

```
class CfdWorkbench(Workbench):
"CFD workbench object"
def __init__(self):
#...
    import CfdPreferencePage
    FreeCADGui.addPreferencePage(CfdPreferencePage.CfdPreferencePage, "CFD")
```

3. Icons of the Preference

```
from PySide import QtCore
ICONS_PATH = CfdTools.getModulePath() + "/Resources/icons"
QtCore.QDir.addSearchPath("icons", ICONS_PATH)
```

“preferences-cfd.png” or “preferences-cfd.svg” must be in icon search path The icon file name pattern “preferences-”

Chapter 9

Developing CFD Module Based on FemWorkbench

9.1 Design of CFD solver for FreeCAD

9.1.1 Adding CFD analysis to FreeCAD

Solidworks provides not only FEM function, but also CFD function. See the SolidWorks flow-simulation. It is desirable that FreeCAD can have such a feature.

Instead of creating a new CFD or CAE module, I am trying to add CFD function to the the current Fem workbench and reuse most of the infrastructure.

See Appendix FreeCAD From Fem workbench towards a full-fledged CAE workbench

CFD simulation needs a more complex setup and dedicated mesh, thereby, in FreeCAD engineering an accurate simulation is not the design aim. Importing a FreeCAD model into other pre-processing tools for meshing and tweaking the experiment setup many times, is needed for serious study.

9.1.2 Literature Review

OpenFoam is not the only free open source CFD solver, but it is powerful. A free GUI case setup tool is missing (arguably).

It is not designed for windows, but usable via **Cygwin**: see **FreeFoam**. It is possible to add Cygwin to the PATH as `C:\cygwin\bin`, then run the solver from command line. Furthermore, it can be run in a container, or even the Ubuntu-on-Windows subsystem as in Windows 10.

Requirement anslysis: see appendix FreeCAD combining the strength of FreeCAD and Salome

Free Solver selection: External solver, it can potentially use a solver of any license.

9.1.3 Roadmap

- Current limitation: FreeCAD FEM is designed only for MechanicalAnalysis and the Solver is tightly coupled with analysis object, not pluggable design. `JobControlTaskView` should be reusable by CFD solver after some refactoring work.
- Case writer is the primary task for function of CFD simulation
- FemMesh export into UNV format, but it does not export boundary condition.
- Only Solid mechanical material is defined in Material module, but no fluid material.
- `BoundaryCondition` for CFD is not defined, this could be derived from `Fem::Constraint`
- View result back to FreeCAD is highly challenging task, thus external

9.1.4 Python or cplusplus ?

It is possible to extend function of `DocumentObject` and `ViewProvider` in Python. The Howto and limitation of developing modules in python has been discussed in the previous chapters.

Example code for type checking in cpp:

```
(obj->getTypeId().isDerivedFrom(Fem::FemSolverObject::getClassTypeId()))

assert analysis_obj.TypeId == "Fem::FemAnalysisPython"

analysis_obj.isDerivedFrom('')
```

TypeId is string representation
documentObj.Name is binary representation
Label is unicode string

9.1.5 Documentation and the Wiki

Detailed documentation and excellent code are equally important for the survival of Open Source projects. Thereby, users can enjoy a potentially smoother functioning program and new developers could be facilitated to extend/maintain the source code with greater ease.

For a workbench developer, it is good to have a page on FreeCAD that introduces ways to easily add features, documentation, and make progress undertaking this effort.

Ways to accomplish this:
First of all, write a good in-source documentation and detailed README.md in the git repo.
Secondly, publish the workbench on the FreeCAD wiki

1. Apply for a wiki account via forum private message, see How to get wiki editing permissions
2. Once approved, read this before editing wiki pages
3. Add an item to the external workbenches section, not directly to user hub, which is for official (core) workbench
4. Add one unique page for the workbench itself.

9.2 Design of Cfd Workbench

In this section, developing a workbench in python and extending c++ defined class is explained, for example, extending `FemSolverObject` into solver specific Python classes.

9.2.1 Code re-usage from FEM workbench

- Fluid material is supported via `FemMaterial` in Fem Module
  *`MechanicalMaterial.py` in Fem is refactored into a general material object for CAE. Material properties are grouped into mechanical, fluidic, and electromagnetic. See forum discussionTODO. CfdWorkbench
- FemMesh and Netgen and Gmsh meshing taskpanel can be imported directly into CFD workbench
- VTK pipeline for post-processing

9.2.2 C++ related code are committed directly to Fem module

- `FemSolverObject`: abstract `DocumentObject` for any CAE solver; concrete solver is developed in C++
- `FemConstraintFluidBoundary`: a single type to represent all boundary conditions in CFD: pressure, velocity, turbulence, and thermal
- **Note**: see the later section on design and implementation of `FemConstraintFluidBoundary` in c++
- VTK mesh import and export
- CFD results exported as VTK file format for building up VTK pipe

9.2.3 Init.py and InitGui.py

This is a pure python module/workbench. A template of empty workbench could be downloaded from Bernd' git:

- Load commands into workbench, which will load new python module as in cpp mode: `src/Mod/Fem/Gui/AppFemGui.cpp`
- Add MenuItem and Toolbar items for this module
- Resource (icon files)
- Translation
- Example and testing
- CMake setting (module `CMakelists.txt` and a global option to activate CFD workbench)

As an addon module made in pure Python, no module `CMakeLists.txt` is needed. Just download this module folder 'Cfd' into `~/.FreeCAD/Mod/` or the FreeCAD installation `Mod/` subfolder.

9.2.4 CfdAnalysis.py

- `makeCfdAnalysis()` creates a `FemAnalysis` object within the current document, defined in `CfdAnalysis.py`
- No need to Extend `Fem::FemAnalysisObject` into `CfdAnalysis` class in python
- `ViewProviderCfdAnalysis` python class is necessary as double-click will activate CFD workbench

9.2.5 CfdTools.py: utility and mesh export

UNV to foam, mesh renumbering, thereby, a result mesh is needed to show Result

https://github.com/OpenFOAM/OpenFOAM-2.2.x/blob/master/applications/utilities/mesh/conversion/ideasUnvToFoam/ideasUnvToFoam.C

9.2.6 CfdRunnable.py: solver specific runner

This class and its derived, equal to FemTools.py family, hides solver specific implementation. Thereby, `TaskPanelCfdSolverControl` can be shared by any CFD solver. The Cfd runnable write solver input file, run the solving process and finally load the result back to FreeCAD.

9.2.7 FoamCaseWriter: write OpenFOAM case setting files

This class extracts information from FreeCAD GUI for `FoamCaseBuilder`, e.g. mesh, material, solver setup and boundary, while the actual case builder is done by `FoamCaseBuilder`

9.2.8 FoamCaseBuilder: a python module to build OpenFOAM case

This is an independent python module, it will be developed in parallel with FreeCAD CFD workbench

- Export UNV mesh with boundary conditions FaceSet
- Case setup by setting boundary condition in workbench
- Case build up from scratch by generating OpenFOAM case setting files
- Case check or update case setup
- `TestBuilder.py` shows a tutorial about how to build a case via script once mesh file is ready

9.2.9 CfdResult: to view result in FreeCAD

- `CfdResult.py`: This class only defines properties representing CFD result, pressure, velocity, temperature, etc.

It is extended from the c++ class: `FemResultObject`, shared by any CFD solver.

- `CfdResultFoamVTK.py`: load results from OpenFOAM solver

OpenFOAM result is exported in VTK legacy file, then read by python-vtk6 module to show as `FemResultObject` in FreeCAD. Only scalers like pressure can be illustrated as different color in `FemMesh` nodes. Velocity vector will not be supported although `FemPostPipeline` is a promising solution.

This module will be re-implemented in c++ to save computation time, since CFD meshes are always huge.

- `TaskPanelCfdResult.py`: select scalar to be shown as colormap on `FemMesh` via modifying `ViewProviderFemMesh` properties

---

9.3 Create FemSovlerObject

9.3.1 Why FemSolverObject is needed?

The Solver class provide information for QProcess to start external solver. It is mainly designed for CFD for the moment, but any solver like Fem, could use it. Another commandline property could be added, or built from current property,so JobControlTaskPanel will be reused by renaming Calculix (QProcess Object) -> SolverProcessObject or like name. Although ccx works perfect now, we are not locked to only one Fem solver.

Solver should be pluggable, swappable. Analysis is a pure container (DocumentObjectGroup) to search for Mesh and Solver Object, from my perspective. Currently, some properties are added into AnalysisObjects, but in Salome or Ansys workbench, Solver is an object equal to Mesh. A lot of parameters, switches are needed to tweak solver, they are not belong to Analysis,but solver specific.

Define a SolverObject can do persistence and replay of solver setup, and work without GUI. SolverObject can be subclass in python to deal with specific solver.

9.3.2 App::DocumentObject derived class: FemSovlerObject.h

[src/Mod/Fem/App/FemResultObject.h], [src/Mod/Fem/App/FemResultObject.cpp] are good templates for this new feature. Just copying and replacing, we are ready to make our own DocumentObject/Feature.

Different from *Fem::ConstraintFluidBoundary*, *Fem::SolverObject* has defined `Fem::SolverObjectPython` and `FemGui::ViewProviderSolverPython` via `FeaturePythonT` for convenient extension by python. DocumentObject for each specific solver, can store solver specific properties. ViewProvider for Each specific solver, e.g. `_ViewProviderFemSolverCalculix.py`, via proxy of `FemGui::ViewProviderSolverPython`, overrides the double-click event to bring up the TaskPanel coded in Python,i.e. `_TaskPanelFemSolverCalculix.py`.

How will FemSolver's properties be visible to Python ? [src/Mod/Fem/App/FemResultObject.h] is a good example, it contains App::Property and has corresponding python class: [src/Mod/Fem/FemResultObjectTaskPanel.py] DocumentObject's properties can be accessed in GUI, property view in combi view.

src/Mod/Fem/App/FemSolverObject.cpp

```cpp
#include <App/FeaturePythonPyImp.h>
#include <App/DocumentObjectPy.h>
...
namespace App {
/// @cond DOXERR
PROPERTY_SOURCE_TEMPLATE(Fem::FemSolverObjectPython, Fem::FemSolverObject)
template<> const char* Fem::FemSolverObjectPython::getViewProviderName(void) const {
    return "FemGui::ViewProviderSolverPython";
}
template<> PyObject* Fem::FemSolverObjectPython::getPyObject(void) {
    if (PythonObject.is(Py::_None())) {
        // ref counter is set to 1
        PythonObject = Py::Object(new App::FeaturePythonPyT<App::DocumentObjectPy>(this),true);
    }
    return Py::new_reference_to(PythonObject);
}
```

```
// explicit template instantiation
template class AppFemExport FeaturePythonT<Fem::FemSolverObject>;
```

- In the App folder, copy *FemResultObject.h* and *FemResultObject.cpp* into FemSovlerObject.h and FemSovlerObject.cpp
- Replace all occurrence of "ResultObject" with "SolverObject" in FemSovlerObject.h and FemSovlerObject.cpp

ViewProvider type must agree with definition in

```
getViewProviderName(void) const {
    return "FemGui::ViewProviderSolverPython";
}
```

- Add some Properties into this FemSovlerObject class derived from DocumentObject if necessary

ADD_PROPERTY_TYPE macro function is defined in src/App/PropertyContainer.h

`ADD_PROPERTY(_prop_, _defaultval_) , ADD_PROPERTY_TYPE(_prop_, _defaultval_, _group_,_type_,_Docu_)`

It is decided to add all properties in python, thereby, c++ class has no properties in cpp.

- Add type initialisation and header inclusion into *FemApp.cpp* for both cpp and python types

```
#include "FemSolverObject.h"
...
    Fem::FemSolverObject            ::init();
    Fem::FemSolverObjectPython      ::init();
```

- add these 2 files into in *App/CMakeList.txt*

9.3.3 Gui part: ViewProviderSolver

- in Gui folder, copy `ViewProviderResult.h` and `ViewProviderResult.cpp` into `ViewProviderSolver.h` and `ViewProviderSolver.cpp`
- replace all occurrence of "Result" with "Solver" in ViewProviderSolver.h and ViewProviderSolver.cpp
- no special render in 3D viewer for ViewProviderSolver class, derived from ViewProvider
- Make sure this object can be dragged into FemAnalysis FemAnalysis is derived from DocumentObjectGroup see [src/Mod/Fem/Gui/ViewProviderFemAnalysis.cpp] `bool ViewProviderFemAnalysis::canDragObject(App::DocumentObject obj) const`

  ```
  #include "FemSolverObject.h"
  ...
  else if (obj->getTypeId().isDerivedFrom(Fem::FemSolverObject::getClassTypeId()))
      return true;
  ```

- taskview to be coded in python to edit solver property and run the solver like "AnalysisType", "CaseName", etc.
- add type initialisation and header inclusion into *FemGuiApp.cpp*

```
#include "ViewProviderSolver.h"
...
    FemGui::ViewProviderSolver                  ::init();
    FemGui::ViewProviderSolverPython            ::init();
```

- add these 2 files into in *Gui/CMakeList.txt*

9.3.4 Command to add FemSolver to FemWorkbench

This section does not reflect the current code condition

FemSolverObject will be extended in python, hence toolbar or menuItem are only created for specific solver like Calculix Solver. The following code is only a demo if implemented in cpp.

- add MenuItem and ToolBarItem to FemWorkbench (should be removed) by adding new class FemSolverCommand. src/Mod/Fem/Gui/Command.cpp the closest command class is *CmdFemConstraintBearing*

```
#include <Mod/Fem/App/FemSolverObject.h>

DEF_STD_CMD_A(CmdFemCreateSolver);
...
```

- add cmd class into workbench

```
void CreateFemCommands(void){
{
    Gui::CommandManager &rcCmdMgr = Gui::Application::Instance->commandManager();
    ...
    rcCmdMgr.addCommand(new CmdFemCreateSolver());
```

- src/Mod/Fem/Gui/Workbench.cpp

```
Gui::ToolBarItem* Workbench::setupToolBars() const
{
    Gui::ToolBarItem* root = StdWorkbench::setupToolBars();
    Gui::ToolBarItem* fem = new Gui::ToolBarItem(root);
    fem->setCommand("FEM");
    ...
         << "Fem_CreateSolver"
   ...
Gui::MenuItem* Workbench::setupMenuBar() const
{
    Gui::MenuItem* root = StdWorkbench::setupMenuBar();
    Gui::MenuItem* item = root->findItem("&Windows");
    Gui::MenuItem* fem = new Gui::MenuItem;
    root->insertItem(item, fem);
    fem->setCommand("&FEM");
    ...
         << "Fem_CreateSolver"
```

- add new SVG icon file "fem-solver.svg" in Gui/Resource
- add "fem-solver.svg" file into Fem.qrc XML file https://doc.qt.io/qt-5/resources.html resource icon images are built into bindary file FemGui.so or FemGui.dll. cmake has one line to rebuilt resources.
- add or update Translation. This is temporarily left behind, until the code is stable.
- `git add <the above newly added file>` If you forget to work in a branch, you can `git stash branch testchanges` see https://git-scm.com/book/en/v1/Git-Tools-Stashing

---

9.4 Boundary condition settings for CFD

9.4.1 Design of FemConstraintFluidBoundary

Class `Fem::ConstraintFluidBoundary` should be derived from `FemConstraint` and adapted from some concrete class like `FemConstraintFixed`, to reduce the work. As python has limitation, e.g. Coin3D scene, there must be coded in C++. The closest class is `FemConstraintForce`, which is derived from `FemConstraint`, except no PythonFeature, but adding TaskPanel.

Modelled after CFX, a commercial CFD tool, boundary conditions are grouped into 5 categories, inlet, outlet, symmetry, wall, opening (freestream/far field in other tools). `BoundaryType` Combobox is used to select from the categories. For each categories, there is another combobox for `Subtype`, e.g. inlet and outlet has different valueType: pressure, flowrate, velocity, etc. A task panel containing properties like: `Value, Direction, Reversed`, could be hidden if no value is needed for any specific boundary subtype.

"Symmetry" should be named more generally as "interface", which can be any special boundaries: wedge(axisymmetry), empty(front and back face for 2D domain, single layer 3D mesh), coupled(FSI coupling interface), symmetry, interior (baffle), processor(interface for domain decomposition), cyclic (Enables two patches to be treated as if they are physically connected), etc.

inlet {totalPressure, velocity, flowrate} outlet {pressure, velocity, inletOutlet} wall {fixed, moving, slip} freestream {freestream} interface {empty, symmetry, cyclic, wedge}

Only uniform value boundary type is supported in GUI, user should edit the case file for OpenFOAM supported csv or function object non-uniform boundary.

The turbulent inlet and thermal boundary condition is editable in the tab of boundary condition, which is accessed by tab in boundary control panel

Other solver control, like gravity, reference pressure, is the internal field initialisation/body force for pressure and velocity.

9.4.2 procedure of adding ConstraintFluidBoundary class

- DocumentObject `Fem::ConstraintFluidBoundary`
- add file names into `App/CMakeList.txt`
- type initialisaton `App/FemApp.cpp`
- ViewProvider `FemGui::ViewProviderConstraintFluidBoundary`
- TaskPanel and ui “
- add file names `Gui/CMakeList.txt`
- type initialisaton `Gui/FemGuiApp.cpp`
- add svg icon file and update XML resource file `Fem.qrc`
- add menuItem in FemWorkbench <Gui/Command.cpp> and <Gui/Workbench.cpp>

9.4.3 FemConstraintFluidBoundary.h and FemConstraintFluidBoundary.cpp

replace name “FemContraintForce” -> “FemConstraintFluidBoundary”

Add new properties “BoundaryType”,“Subtype”, etc, in corresponding header and cpp files

```
#include <App/PropertyStandard.h>
...
    App::PropertyEnum BoundaryType;
    App::PropertyEnum Subtype;
    App::PropertyFloat BoundaryValue; // rename "Force" into "BoundaryValue"
```

Following code should be added to function `void ConstraintFluidBoundary::onChanged(const App::Property* prop)` to update the Subtype enums in property editor once BoundaryType is changed.

```
    if (prop == &BoundaryType) {
        std::string boundaryType = prop.getValueAsString();
        if (boundaryType == "wall")
        {
            Subtype.setEnums(WallSubtypes);
        }
        else if (boundaryType == "interface")
        {
            Subtype.setEnums(InterfaceSubtypes);
        }
        else if (boundaryType == "freestream")
        {
            Subtype.setEnums(FreestreamSubtypes);
        }
        else if(boundaryType == "inlet" || boundaryType == "outlet")
        {
            Subtype.setEnums(InletSubtypes);
        }
        else
        {
            Base::Console().Message("Error: this boundaryType is not defined\n");
        }
    }
```

Definition of the Macro:

```
ADD_PROPERTY_TYPE(BoundaryType,(1),"ConstraintFluidBoundary",(App::PropertyType)(App::Prop_None),
              "Basic boundary type like inlet, wall, outlet,etc");
```

This a macro function, some parameter must be embraced with parenthesis, like varialbe with namespace `(App::Prop_None)`. The second parameter can not be zero `(0)`.

For `setEnums(const char**)` at least 2 enums are needed, e.g. `subtypes = {"A", "B", NULL};`

register `Fem::ConstraintFluidBoundary` and type `init()` in [src/Mod/Fem/App/FemApp.cpp]

9.4.4 ViewProviderConstraintFluidBoundary.h

(changed combobox type should trigger a redraw) Only outlet, will show arrow as `FemConstrainForce`, inlet has the arrow but in reverse direction (flow into the geometry) Other boundary types will shows as `FemConstrainFixed`. However, simply merging codes of two viewProviders into `ViewProviderFemConstraintFluidBoundary.cpp` does not work properly.

`void ViewProviderFemConstraintFluidBoundary::updateData(const App::Property* prop)` only update property data, while actural drawing is done in base class method: `ViewProviderFemConstraint::updateData(prop);`

```
//change color to distinguish diff subtype
App::PropertyColor          FaceColor;

// comment out *createCone* will make draw "interface" type and "freestream" type of fluid boundary
void ViewProviderFemConstraint::createFixed(SoSeparator* sep, const double height, const double width, const b
{
    createCone(sep, height-width/4, height-width/4);
    createPlacement(sep, SbVec3f(0, -(height-width/4)/2-width/8 - (gap ? 1.0 : 0.1) * width/8, 0), SbRotation(
    createCube(sep, width, width, width/4);
}
```

adding header and init function into src/Mod/Fem/Gui/AppFemGui.cpp This module is not designed to be extended in python as other FemConstraint class, thereby only cpp type are declared.

```
#include "ViewProviderFemConstraintFluidBoundary.h"
...
PyMODINIT_FUNC initFemGui()
{
    FemGui::ViewProviderFemConstraintFluidBoundary          ::init();
```

9.4.5 TaskFemConstraintFluidBoundary

- copy from nearest file to create: `TaskFemConstraintFluidBoundary.h` and `TaskFemConstraintFluidBoundary.cpp`,
- getters for newly added properties should be added. e.g. `double getForce(void) const;` is replaced with `getBoundaryType(void) const`
- property changed signal slots: `void onForceChanged(double);` slot is replaced, and `bool TaskDlgFemConstraintFluidBounda`
  "Force" is replaced by "BoundaryValue"

Note: event will not fired, if wrong slot function signature is specified in connection()

Only face can be selected as fluid boundary, via removing edge selection in `void TaskFemConstraintFluidBoundary::onSelectionCha`
`Gui::SelectionChanges& msg)`

`TaskFemConstraintFluidBoundary.ui` when create new ui file from an existent ui file, make sure the toplevel object name is also properly renamed in Qdesigner. Event is not defined in this ui file, but src/Mod/Fem/Gui/TaskFemConstraintFluidBoundary.cpp

9.4.6 svg icon "fem-fluid-boundary" and update Fem.qrc,

I use **inkscape** to make new svg icon for this class and add file name into src/Mod/Fem/Gui/Resources/Fem.qrc

9.4.7 GUI menubar and toolbar: Command.cpp and Workbench.cpp

in src/Mod/Fem/Gui/Command.cpp file, a new command class *CmdFemConstraintFluidBoundary*. Also `rcCmdMgr.addCommand(new CmdFemConstraintFluidBoundary());` must be added in in `void CreateFemCommands(void)` at the end of this file. Otherwise, error “Unknown command ‘Fem_ConstraintFluidBoundary’ ” will print in your Console.

in src/Mod/Fem/Gui/Workbench.cpp both Toolbar and MenuBar should be updated, otherwise the GUI will not shown up.

9.5 Example of extending FemSolverObject in python

9.5.1 Procedure for extending FeaturePythonT object

`CfdSolverFoam.py  _FemSolverCalculix.py`

- add dialog UI into update property of `FemSolverObject`
- design `TaskPanelCfdSolverControl.ui` dialog GUI form by QtDesigner
- add `_TaskPanelCfdSolverControl` python class
- add `ViewProviderCfdSolverFoam python` class
- Macro replay/ document import should work now.

update CMakeList.txt and resource

- add new files into in *Gui/CMakeList.txt*
- deprecated class _FemAnalysis _ViewProviderFemAnalysis (feature dropped)
- rename and refactoring of _JobControlTaskPanel.py (feature dropped)
- create new icons file

9.5.2 code for extending FeaturePythonT object

`makeCfdSolverFoam` is the magic connection between cpp class and Python class. It returns a document object derived type “Fem::FemSolverObjectPython”, which is defined in c++ using FeatureT template. Extra properties can be added by `CfdSolverFoam(obj)` constructor. Furthermore, ViewProvider can be extended by `_ViewProviderCfdSolverFoam` python class.

```
def makeCfdSolverFoam(name="OpenFOAM"):
    obj = FreeCAD.ActiveDocument.addObject("Fem::FemSolverObjectPython", name)
    CfdSolverFoam(obj)
    if FreeCAD.GuiUp:
        from _ViewProviderCfdSolverFoam import _ViewProviderCfdSolverFoam
        _ViewProviderCfdSolverFoam(obj.ViewObject)
    return obj
```

CfdSolver is a generic class for any CFD solver, defining shared properties

```
class CfdSolver(object):
    def __init__(self, obj):
        self.Type = "CfdSolver"
        self.Object = obj  # keep a ref to the DocObj for nonGui usage
        obj.Proxy = self  # link between Fem::FemSolverObjectPython to this python object

    # API: addProperty(self,type,name='',group='',doc='',attr=0,readonly=False,hidden=False)
        obj.addProperty("App::PropertyEnumeration", "TurbulenceModel", "CFD",
                        "Laminar,KE,KW,LES,etc")
        obj.TurbulenceModel = list(supported_turbulence_models)
        obj.TurbulenceModel = "laminar"
```

OpenFOAM specific properties go into `CfdSolverFoam`

```
class CfdSolverFoam(CfdSolver.CfdSolver):
    def __init__(self, obj):
        super(CfdSolverFoam, self).__init__(obj)
        self.Type = "CfdSolverFoam"
```

`_ViewProviderCfdSolverFoam` is needed to double click and bring up a TaskPanel Although "_TaskPanelCfdSolverControl" can be shared by any cfd solver, but each CFD solver needs a solver-specific CfdRunner

_ViewProviderCfdSolverFoam.py

```
import FreeCAD
import FreeCADGui
import FemGui

class _ViewProviderCfdSolverFoam:
    """A View Provider for the Solver object, base class for all derived solver
    derived solver should implement  a specific TaskPanel and set up solver and override setEdit()
    """

    def __init__(self, vobj):
        vobj.Proxy = self

    def getIcon(self):
        """after load from FCStd file, self.icon does not exist, return constant path instead"""
        return ":/icons/fem-solver.svg"

    def attach(self, vobj):
        self.ViewObject = vobj
        self.Object = vobj.Object

    def updateData(self, obj, prop):
        return

    def onChanged(self, vobj, prop):
        return

    def doubleClicked(self, vobj):
        if FreeCADGui.activeWorkbench().name() != 'CfdWorkbench':
            FreeCADGui.activateWorkbench("CfdWorkbench")
        doc = FreeCADGui.getDocument(vobj.Object.Document)
        if not doc.getInEdit():
            # may be go the other way around and just activate the analysis the user has doubleClicked on ?!
            if FemGui.getActiveAnalysis():
                if FemGui.getActiveAnalysis().Document is FreeCAD.ActiveDocument:
                    if self.Object in FemGui.getActiveAnalysis().Member:
                        doc.setEdit(vobj.Object.Name)
                    else:
                        FreeCAD.Console.PrintError('Activate the analysis this solver belongs to!\n')
                else:
                    FreeCAD.Console.PrintError('Active Analysis is not in active Document!\n')
            else:
                FreeCAD.Console.PrintError('No active Analysis found!\n')
        else:
            FreeCAD.Console.PrintError('Active Task Dialog found! Please close this one first!\n')
        return True

    def setEdit(self, vobj, mode):
        if FemGui.getActiveAnalysis():
            from CfdRunnableFoam import CfdRunnableFoam
            foamRunnable = CfdRunnableFoam(FemGui.getActiveAnalysis(), self.Object)
            from _TaskPanelCfdSolverControl import _TaskPanelCfdSolverControl
            taskd = _TaskPanelCfdSolverControl(foamRunnable)
            taskd.obj = vobj.Object

            FreeCADGui.Control.showDialog(taskd)
        return True

    def unsetEdit(self, vobj, mode):
        FreeCADGui.Control.closeDialog()
        return

    def __getstate__(self):
        return None

    def __setstate__(self, state):
        return None
```

==

9.6 Develop a new CFD solver within CFD workbench

https://github.com/qingfengxia/Cfd This workbench is designed to fit in more solvers. To solve CFD problem in other solvers, you may reuse some of the code in CfdWorkbench, which is a split from FemWorkbench. FluidMaterial.py has several implementation, currently needs a unification. FemConstraintFluidBoundary.cpp can be reused for common CFD boundary types

For example (not yet implemented)

- CfdSolverElmer.py
  derived from CfdSolver which defined most of setting for CFD, include elmer specific settings, example are CfdSolverFoam.py

- CfdCaseWriterElmer.py
  expose only a write_case() method. write_mesh (which should be similar as Bernd has done)and write boundary condition

- CfdRunnableElmer.py
  call the calculation and retrieve the result and display, CfdSolverControl task panel can be reused with tiny modification

- then add _CommandCfdSolverElmer.py _ViewProviderCfdSolverElmer.py, icon svg file can be adapted from *Foam.py add into CfdWorkbench via InitGui.py

+CfdResultVTKElmer.py
load result . currently this piece of code (CfdResult and taskpanel) is under review, there is no need to

Chapter 10

Testing and Debugging Module

10.0.1 Python and c++ IDE

For python coding, a text editor with grammar highlight + QtDesigner is enough to code in the first case. QtDesigner can be used to generate and edit Qt ui files. Spyder, which comes with Anaconda, is a good and lightweight Python IDE with debugger support.

Various c++ IDE are available to support Cmake project. Visual Studio 2015 is essential for development on Windows. Cmake project can be mapped to a VS solution (*.sln) with a bundle of projects corresponding to

Latest QtCreator works with Qt 4.x and Qt 5.x; it also support CMake project.

see tutorial of import cmakelists.txt as project file http://doc.qt.io/qtcreator/creator-project-cmake.html

1. Select File > Open File or Project.
2. Select the CMakeLists.txt file from your CMake project.
3. Select out of source build folder, you may have already
4. Run camke once to build up pcb database
5. The whole FreeCAD source tree will be loaded into QtCreator
6. Further configuration 'Projects > Build Settings > Edit build configuration'

QtCreator or similar IDE can provide:

- graphic git support
- code completion and API hint
- fast code navigation by 'goto definition'
- break point setup in debug
- CMakeLists.txt editor with grammar highlight and keyword completion, etc.

10.1 Extra tools for module developer

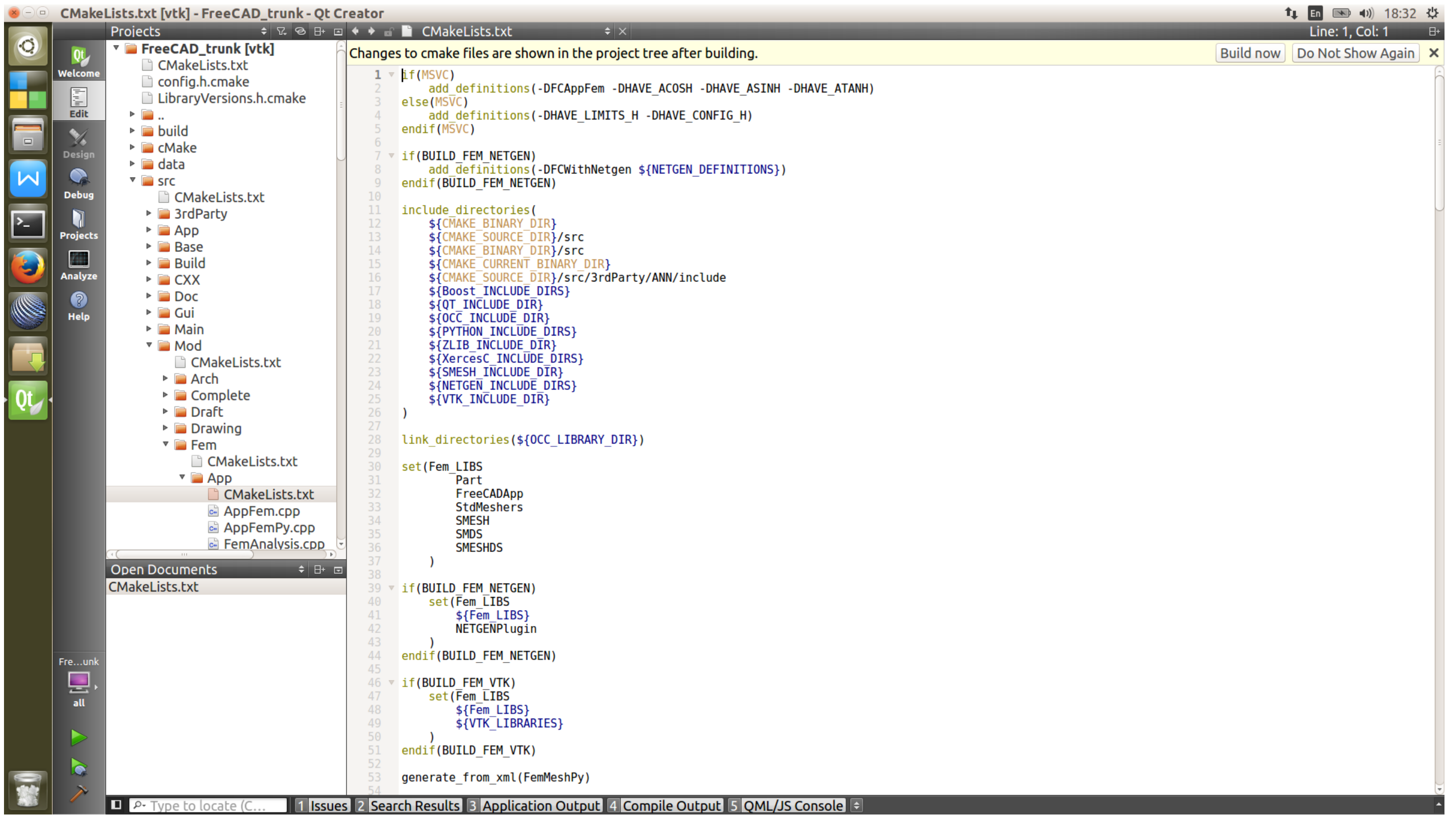
- InkScape to generate SVG icon Great vector drawing program. Adhers to the SVG standard and is used to draw Icons and Pictures. Get it at https://www.inkscape.org

- Doxygen to generate documentation A very good and stable tool to generate source documentation from the .h and .cpp files.

- Gimp to edit XPM icon file Not much to say about the Gnu Image Manipulation Program. Besides it can handle .xpm files which is a very convenient way to handle Icons in QT Programs. XPM is basically C-Code which can be compiled into a program. Get the GIMP here: https://www.gimp.org

- ccache to reduce building time

- cppcheck to improve coding quality



Changes to cmake files are shown in the project tree after building.

```
if(MSVC)
    add_definitions(-DFCAppFem -DHAVE_ACOSH -DHAVE_ASINH -DHAVE_ATANH)
else(MSVC)
    add_definitions(-DHAVE_LIMITS_H -DHAVE_CONFIG_H)
endif(MSVC)

if(BUILD_FEM_NETGEN)
    add_definitions(-DFCWithNetgen ${NETGEN_DEFINITIONS})
endif(BUILD_FEM_NETGEN)

include_directories(
    ${CMAKE_BINARY_DIR}
    ${CMAKE_SOURCE_DIR}/src
    ${CMAKE_BINARY_DIR}/src
    ${CMAKE_CURRENT_BINARY_DIR}
    ${CMAKE_SOURCE_DIR}/src/3rdParty/ANN/include
    ${Boost_INCLUDE_DIRS}
    ${QT_INCLUDE_DIR}
    ${OCC_INCLUDE_DIR}
    ${PYTHON_INCLUDE_DIRS}
    ${ZLIB_INCLUDE_DIR}
    ${XercesC_INCLUDE_DIRS}
    ${SMESH_INCLUDE_DIR}
    ${NETGEN_INCLUDE_DIRS}
    ${VTK_INCLUDE_DIR}
)

link_directories(${OCC_LIBRARY_DIR})

set(Fem_LIBS
        Part
        FreeCADApp
        StdMeshers
        SMESH
        SMDS
        SMESHDS
    )

if(BUILD_FEM_NETGEN)
    set(Fem_LIBS
        ${Fem_LIBS}
        NETGENPlugin
    )
endif(BUILD_FEM_NETGEN)

if(BUILD_FEM_VTK)
    set(Fem_LIBS
        ${Fem_LIBS}
        ${VTK_LIBRARIES}
    )
endif(BUILD_FEM_VTK)

generate_from_xml(FemMeshPy)
```

Figure 10.1: Load FreeCAD source tree to QtCreator

10.1.1 ccache to accelerate compilation

install `ccache` and rerun the `cmake` to https://www.virag.si/2015/07/use-ccache-with-cmake-for-faster-compilation/

for a cmake project, add the following code into top level cmakelists.txt

```
# Configure CCache if available
find_program(CCACHE_FOUND ccache)
if(CCACHE_FOUND)
        set_property(GLOBAL PROPERTY RULE_LAUNCH_COMPILE ccache)
        set_property(GLOBAL PROPERTY RULE_LAUNCH_LINK ccache)
endif(CCACHE_FOUND)
```

10.2 C++ debugging

10.2.1 Build FreeCAD from source

First of all, make sure you can build the official source once done `git clone` which confirms you can install all the lib dependent.

compiler's warning is the first place to spot error and potential bugs.

10.2.2 Reduce compiling time by ccache

10.2.2.1 Incremental compilation by ccache

10.2.2.2 Update *.ui file

10.2.2.3 Compile only one module

> after changing an ui-file like this one (https://github.com/FreeCAD/FreeCAD/blob … ces-ifc.ui) I have to run make clean && make to get the changes active.
>
> Re: make clean after changing an *.ui file Postby wmayer » Thu Aug 06, 2015 4:46 pm

In this case `cd` into the Arch directory first before running "make clean" because then it only rebuilds this module and not the whole project.

10.2.3 Print debug info

Qt debug http://doc.qt.io/qt-4.8/debug.html

The console class This class manage all the stdio stuff. here is the generated document for src/Base/Console.h

> This includes Messages, Warnings, Log entries and Errors. The incoming Messages are distributed with the FCConsoleObserver. The FCConsole class itself makes no IO, it's more like a manager. ConsoleSingleton is a singleton! That means you can access the only instance of the class from every where in c++ by simply using:

```
#include <Base/Console.h>
//...
Base::Console().Log("Stage: %d",i);
```

src/Base/Tools.h

```
struct BaseExport Tools
{
    /**
     * @brief toStdString Convert a QString into a UTF-8 encoded std::string.
     * @param s String to convert.
     * @return A std::string encoded as UTF-8.
     */
    static inline std::string toStdString(const QString& s)
        { QByteArray tmp = s.toUtf8(); return std::string(tmp.constData(), tmp.size()); }

    /**
     * @brief fromStdString Convert a std::string encoded as UTF-8 into a QString.
     * @param s std::string, expected to be UTF-8 encoded.
     * @return String represented as a QString.
     */
    static inline QString fromStdString(const std::string & s)
        { return QString::fromUtf8(s.c_str(), s.size()); }
}
```

Example usage of QString from *std::string*, `#include <Base/Tools.h> Base::Tools::fromStdString()`

10.2.4 Tips for debugging cpp code

- compile only the module `/opt/FreeCAD/build/src/Mod/Fem$ make`
- show logging info:

```
Edit -> Preferences -> output windows -> record log
```

10.3 Step-by-step debugging via gdb

Once I run into a *Segmentation fault (core dumped)* for "import vtk" in python console of FreeCAD and FreeCADCmd. GDB might tell you more information, like which function, file object caused this fault.

https://sourceware.org/gdb/onlinedocs/gdb/Continuing-and-Stepping.html

Example:

```
gdb freecadcmd
gdb start
gdb continue
gdb where
```

10.3.1 Generating a Backtrace

https://www.freecadweb.org/wiki/Debugging#Generating_a_Backtrace

---

10.4 Python debugging

See also https://www.freecadweb.org/wiki/Debugging#Python_Debugging ### Tips for developing in Python

- remember : modified python file will not take effect until FreeCAD is relauched
- always try to write a test function avoiding test in GUI mode

  Debugging functionality without GUI could be straightforwards, e.g. src/Mod/Fem/TestFem.py,

For Gui functions, recarding all operation into Macro, and replay them can accelerate the testing.

- symbolic link to python files for quick test without installation after compiling for the mixed c++ and python coding

```
ln -s /opt/Cfd/FoamCaseBuilder  fc_build_dir/Mod/Cfd/FoamCaseBuilder
```

10.4.1 Where is python's print message?

`print "Error Message"` does not work in FreeCAD, neither PythonConsole in GUI mode, or terminal starting freecad program (stdout can be viewed in ReportView, by activating this view). By changing the default preference, it is possible to show print message from python module.

- Method 1: FreeCAD.Console.PrintMessage() for show up
- Method 2: Print to TextEdit widget in your specific TaskPanel class

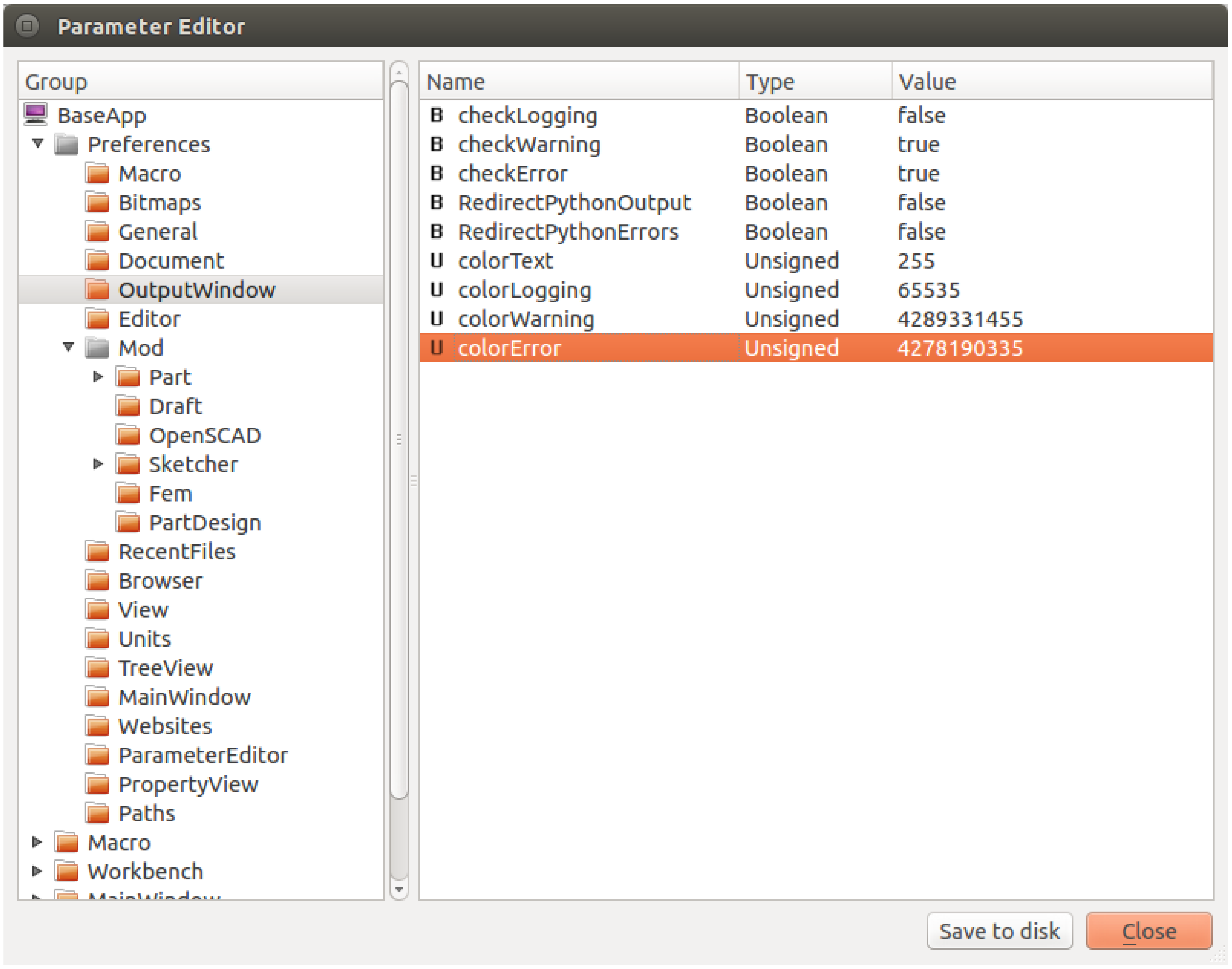
src/Gui/GuiConsole.h

```
/** The console window class  This class opens a console window when instantiated  and redirect the stdio stre
 *  After instantiation it automatically register itself at the FCConsole class and gets all the FCConsoleObse
 */
```

10.4.2 Reload edited python module

Discussion on reload python code without restart FreeCAD

```
>>> import FemTools
>>> FemTools.FemTools.known_analysis_types
['static', 'frequency']
---- Here I added one more analysis type to known_analysis_type in FemTools.py file -----
>>> reload(FemTools)
<module 'FemTools' from '/home/przemo/software/FreeCAD/build/Mod/Fem/FemTools.py'>
>>> FemTools.FemTools.known_analysis_types
['static', 'frequency', 'mock']
```



Name	Type	Value
B checkLogging	Boolean	false
B checkWarning	Boolean	true
B checkError	Boolean	true
B RedirectPythonOutput	Boolean	false
B RedirectPythonErrors	Boolean	false
U colorText	Unsigned	255
U colorLogging	Unsigned	65535
U colorWarning	Unsigned	4289331455
U colorError	Unsigned	4278190335

Figure 10.2: changing the default preference to show print message

This section is an excerpt from *Sri Sanketh Uppalapati* article on GeeksforGeeks

> `reload()` reloads a previously imported module. This is useful if you have edited the module source file using an external editor and want to try out the new version without leaving the Python interpreter. The return value is the module object. Note: The argument should be a module which has been successfully imported.

```
#For Python2.x
reload(module)

#For above 2.x and <=Python3.3, which is not common in 2019

import imp
imp.reload(module)

#For >=Python3.4

import importlib
importlib.reload(module)
```

10.4.3 Test FreeCAD GUI function by python script

We already know that non-GUI code of FreeCAD can be tested without starting GUI, however, the GUI is still needed for testing. It is possible to test GUI code by script, see an example of CFD workbench test script. It can save time to utilize the GUI. Potentially, a `git hook` can be set to run this script for each commit. This is useful since a third-party UnitTest script will not be run by FreeCAD Test workbench.

```
# run a gui test for FreeCAD GUI functions

import sys
sys.path.append('/usr/lib/freecad-daily/lib')

import FreeCAD as App
import FreeCADGui as Gui

Gui.showMainWindow()
Gui.activateWorkbench("StartWorkbench")

##########################################################
App.newDocument("Unnamed")
App.setActiveDocument("Unnamed")
App.ActiveDocument=App.getDocument("Unnamed")
Gui.ActiveDocument=Gui.getDocument("Unnamed")
Gui.activateWorkbench("PartWorkbench")
App.ActiveDocument.addObject("Part::Cylinder","Cylinder")
App.ActiveDocument.ActiveObject.Label = "Cylinder"
App.ActiveDocument.recompute()
Gui.SendMsgToActiveView("ViewFit")

Gui.activateWorkbench("CfdWorkbench")
Gui.ActiveDocument.setEdit('Cylinder',0)

import FemGui
import CfdObjects
CfdObjects.makeCfdAnalysis('CfdAnalysis')
FemGui.setActiveAnalysis(App.activeDocument().ActiveObject)
FemGui.getActiveAnalysis().addObject(CfdObjects.makeCfdSolver('OpenFOAM'))
FemGui.getActiveAnalysis().addObject(CfdObjects.makeCfdFluidMaterial('FluidMaterial'))
mesh_obj = CfdObjects.makeCfdMeshGmsh('Cylinder_Mesh')
mesh_obj.Part = App.ActiveDocument.Cylinder
FemGui.getActiveAnalysis().addObject(mesh_obj)
```

```
# more code copied from interpreter widget, whatever recorded by macro recording

##########################################################
Gui.getMainWindow().close()
#Gui.doCommand('exit()')  # another way to exit
```

10.4.4 Testing FreeCAD Python GUI and CLI via AppImage

The AppImage technology allows to construct and distribute a cross Linux distro executable. It also has the ability to be utilized as a testing tool for python code.

10.4.4.1 Background

As mentioned above, a very convenient aspect of FreeCAD is that a majority of it is built in python and doesn't require to manually compile the code (like c++). Essentially, a python file can be modified and upon reloading the python script (or restarting FreeCAD) those changes will be integrated. A developer can quickly work on the latest FreeCAD release using this technique and an appimage. FYI, the following procedure doesn't modify the environment in any way, in other words, nothing gets installed and no environment paths get modified. All one needs to do is the following:

10.4.4.2 Modifying AppImages

```
<FreeCAD>.AppImage --appimage-extract # Decompresses the AppImage
cd squashfs-root/                     # cd in to the expanded AppImage tree
# (Open your python file(s) in you favorite code editor, modify, and save)
./AppRun                              # Run FreeCAD AppImage in it's current state
```

10.4.4.3 Repackaging AppImages

If you've done the above and now want to re-package the AppImage with your latest changes (useful for several reasons though mainly for having others quickly test your modifications) proceed as follows:

```
cd ..                                   # Make sure we're out of squashfs-root/
wget "https://github.com/AppImage/AppImageKit/releases/download/continuous/appimagetool-x86_64.AppImage"
chmod a+x appimagetool-x86_64.AppImage
./appimagetool-x86_64.AppImage squashfs-root
```

- `wget` is a utility to download remote files
- `chmod` will give the `appimagetool` executable privileges
- `appimagetool` is an AppImage utility that converts an AppDir into a self-mounting filesystem image.

- `appimagetool` will repackage the AppImage and compress it making it ready for distribution

Chapter 11

Contribute code to FreeCAD

11.1 Read first

11.1.1 The official guide for developer

Read this first if you want to write code for FreeCAD Some guide lines for contribute code to FreeCAD Roadmap of FreeCAD: search FreeCAD and roadmap to get the last

Those guidelines are a bit terse, it may be worth of demonstration for new developers.

11.1.2 Read FreeCAD forum and state-of-the-art development

FreeCAD development is very community discussion driven. Many of these discussions occur on the FreeCAD forum.

If you want to contribute new features to FreeCAD, you should always first check if someone else has already had that idea and has completed it or if they are just in the process of implementing it.

Secondly, introduce yourself to the community" by posting your idea onto the forum and hear feedback from the FreeCAD developers and users.

11.1.3 Bug Submission Guideliens

FreeCAD forum: https://forum.freecadweb.org/viewtopic.php?f=3&t=5236&sid=d549b5e9876db90237ad0456a5d83158

If you think you found a bug in FreeCAD, you can help greatly by reporting that bug to the FreeCAD developers on the FreeCAD bug tracker at http://www.freecadweb.org/tracker/. Bugs reported here on the forum can be easily overlooked or missed by developers, while on the tracker they are sure to stay in the queue until a developer looks at it.

However, if well-made bug reports can help a lot and make the problem very easy to solve, bad bug reports can do the opposite: They can make the developer spend many unnecessary hours on the problem. Such bad reports are therefore often treated with the lowest possible priority, and you might not see your bug fixed.

So, it is important to follow these guidelines when submitting a bug on the tracker:

1. Make sure that what you found is a bug. The definition of a bug is this: Something that is not working, but for which you are sure that it should be working (for example it worked in the past, or it is advertised on the FreeCAD documentation wiki). Otherwise, it might be simply that what you want is not implemented, or maybe you are doing something wrong.
2. Discuss the problem on the forum first. Open a new topic here on the forum, in the help section, and ask if what you found is a bug. Other users will surely be able to help you determining what you should do.
3. Make sure your bug can be reproduced. This is the most important rule. If the developer cannot reproduce your problem on his machine, he will not be able to fix it. So you must give an exact step-by-step procedure that the developer must follow to see your bug happen. For example:
    1. Open FreeCAD

2. Press the "create new document" button
3. Switch to the Part workbench
4. Press the "Create cube" button
5. FreeCAD crashes

4. Always include your FreeCAD version information. You can easily copy/paste all the needed information by using the "Copy to clipboard" button in FreeCAD, in menu Help -> About FreeCAD. You can then paste that information in your bug report.

5. Include all possible FreeCAD error messages. When something wrong happens in FreeCAD, error messages are often printed, that can give precious information to the developer about what happened. The two places you should check for such error messages in FreeCAD are the Report window (menu View->Views->Report view), the python console (menu View->Views->Python console). Make sure to include as much error text as possible.

6. If possible, also check the terminal and the FreeCAD log for error messages. A third possible place to look for error messages is the system terminal, which you can get on Linux and Mac systems by opening a terminal, and launching FreeCAD from it, by typing "freecad" . A fourth place to look for error messages is the FreeCAD log. To produce such a log, you must either run FreeCAD from a terminal by running "freecad -l" or change the properties of your FreeCAD launch icon, and use "freecad -l" instead of "freecad". After running and closing (or crashing) FreeCAD, you will find a file called freecad.log in your FreeCAD user directory (/home/youruser/.FreeCAD on linux and mac, C:/users/youruser/ApplicationData/FreeCAD on windows).

7. Stay cool and patient. We all know bugs are annoying, but nobody is here to serve you, developers will work on solving bugs when they have time to do so, and efficient bug fixing depends largely on how helpful people submitting bugs are. By following the above rules, you'll be doing a very good job when submitting your bug, and make sure your bug is treated in the best possible manner. Thanks!

11.2 Develop FreeCAD by git

11.2.1 Learn git

- github cheatsheat This section will explain in details: How you can contribute to FreeCAD project
- git from the bottom up
- The 11 Rules of GitLab Flow link to Chinese translation of The 11 Rules of GitLab Flow
- github tutorials for common jobs
- google if you run into trouble

11.2.2 GUI tools for git

The Author recommended gitKraken, a free, portable and powerful GUI tools. It make the advanced operations like merge, cherry pick easier for new git users.

Note: gitKraken is proprietary but can be used for free if the user is working on Open Source Software/Public repositories.

The best features are:

- one click undo and redo
- visual hints
- intuitive merge for conflict

11.3 Workflow to add new feature to FreeCAD

11.3.1 Setup your git repo and follow official master

Suggestion by Fem module developer Przome:

"leave master branch in your github repo the same as the main master branch of FreeCAD main repo. That will make your life much easier when using"git rebase master" to keep you development branch up to date. "

“fork” the official master of FreeCAD in your webbrowser on gitub

clone the forked git into your PC, `git clone https://github.com/<yourgithubusername>/FreeCAD.git` if you have git clone for your fork, you can just add official as remote `git remote add upstream https://github.com/FreeCAD/FreeCAD.git` check you origin and upstream remote setup `git remote -v`

Example output from my the author’s terminal:

```
origin  https://github.com/qingfengxia/FreeCAD.git (fetch)
origin  https://github.com/qingfengxia/FreeCAD.git (push)
upstream    https://github.com/FreeCAD/FreeCAD.git (fetch)
upstream    https://github.com/FreeCAD/FreeCAD.git (push)
```

Also, keep update the local master with upstream, assuming you have not edited the code the master branch `git pull upstream master`

11.3.2 Implement new functionality in branch

Git Workflow for Feature Branches

In your module folder commit to local storage and push to your fork online

```
git checkout <testbranch>
git add --all .
git commit --am "module name: your comment on this commit"
git push origin <testbranch>
```

Particularly, it is a good practice to lead the commit message with the module name

if you are making change at master branch, do not worry: `git checkout -b new_branch_name` will move all your change into new branch , then you can

```
git add --all .
git commit --am "your comment on this commit"
git push origin <testbranch>
```

11.3.3 Jump between branches

`git checkout some_branch` if you got error like this:

> error: Your local changes to the following files would be overwritten by >checkout: Please, commit your changes or stash them before you can switch branches. Aborting

```
git stash
git checkout test_branch
git stash pop
```

Here is a nice little shortcut:

```
git checkout --merge some_branch
```

11.3.4 Keep branch updated with official master

```
git pull --rebase upstream master
```

Merging an upstream repository into your fork

After working on your local fork for a while, probably, there is confilct merge the master. It is quite challenging for new developer like me. What I did is backup my files working on, which is conflicting with remote master, then merge with remote, finally copy my file back and manually merge the changes. As for module developing, changes are limited into single module, it is not a professional way, but simple way.

“Rebase is your friend, merge commits are your enemy.” More here: http://www.alexefish.com/post/52e5652520a0460016000002

Start by making changes to the feature branch you're doing work on. Let's assume that these changes span a few commits and I want to consolidate them into one commit. The first step involves making sure the master branch is up to date with the destination repo's master branch:

- switch to master branch: `git checkout master`
- ensure our master is up to date: `git pull upstream master`
- With the master branch up to date, we'll use git rebase to consolidate: `git checkout your_branch` `git rebase -i master` That command will show a list of each commit. If there is conflict, trouble is coming. By default, it's a classic rebase: cherry-picking in sequence for every commit in the list. Abort anytime if you are not sure, using `git rebase --abort`.

Uisng merge GUI tool for 2-way merge `git mergetool --tool=meld` each time when there is a conflict. After solving the conflict , `git rebase --continue` again.

```
git checkout A
git rebase   B    # rebase A on top of B
local is B,
remote is A
```

Instead of interactive mode, `git rebase master` will give you a list of conflicts. Graphical merge GUI tool can be used and `git rebase --continue`

11.3.5 Merge with GUI mergetool *meld*

If you start three-pane merging tool (e.g. meld, kdiff3 and most of the others), you usually see **LOCAL on the left (official remote master), merged file in the middle and REMOTE (your dev branch) on the right pane**. It is enough for everyday usage. Edit only the merged file in the middle, otherwise, modification on the left and right will lead to trouble/repeating manually merge conflict many times.

What you don't see is the BASE file (the common ancestor of $LOCAL and $REMOTE), how it looked like before it was changed in any way.

advanced topic

Meld has a hidden 3-way merge feature activated by passing in the 4th parameter:

`meld $LOCAL $BASE $REMOTE $MERGED` The right and left panes are opened in read-only mode, so you can't accidentally merge the wrong way around. The middle pane shows the result of merge.For the conflicts it shows the base version so that you can see all the important bits: original text in the middle, and conflicting modifications at both sides. Finally, when you press the "Save" button, the $MERGED file is written - exactly as expected by git. The ~/.gitconfig file I use contains the following settings:

```
[merge]
tool = mymeld
conflictstyle = diff3
[mergetool "mymeld"]
cmd = meld --diff $BASE $LOCAL --diff $BASE $REMOTE --diff $LOCAL $BASE $REMOTE --output $MERGED
```

this opens meld with 3 tabs, 1st and 2nd tab containing the simple diffs I'm trying to merge, and the 3rd tab, open by default, shows the 3-way merge view.

1) $LOCAL=the file on the branch where you are merging; untouched by the merge process when shown to you
2) $REMOTE=the file on the branch from where you are merging; untouched by the merge process when shown to you
3) $BASE=the common ancestor of $LOCAL and $REMOTE, ie. the point where the two branches started diverting the considered file; untouched by the merge process when shown to you
4) $MERGED=the partially merged file, with conflicts; this is the only file touched by the merge process and, actually, never shown to you in meld

The middle pane show (BASE) initially and it turns/saved into (MERGED) as the result of merging. Make sure you move your feature code (LOCAL) from left to the middle and move upstream updated code from the right pane (REMOTE)

http://stackoverflow.com/questions/11133290/git-merging-using-meld

http://lukas.zapletalovi.com/2012/09/three-way-git-merging-with-meld.html

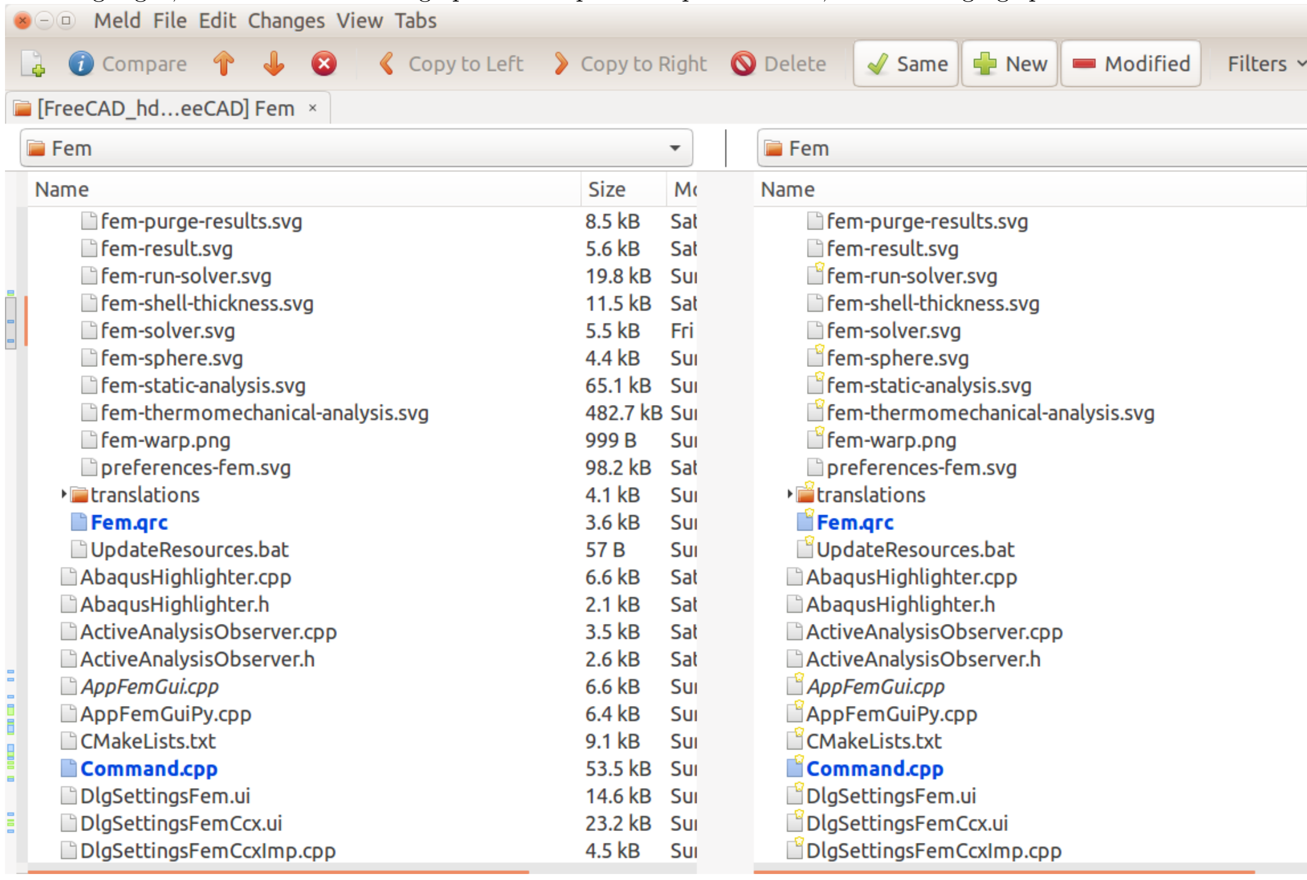
11.3.6 clean branch after failing to rebase

After ignore the official master for half year, I found it is not possible to rebase my feature. It is ended up with kind of merge, instead of smooth playing back my feature commit.

I start to split my feature into C++ section, which is more stable, into a clean branch for pull request. Instead of
 http://meldmerge.org/features.html

```
meld  /opt/FreeCAD_hd/src/Mod/Fem/  /opt/FreeCAD/src/Mod/Fem
```

File with difference will be highlight, double-click will bring up a double-pane comparasion tab, similar merging operaiton is available as in git rebase.



Name	Size	Mo	Name
fem-purge-results.svg	8.5 kB	Sat	fem-purge-results.svg
fem-result.svg	5.6 kB	Sat	fem-result.svg
fem-run-solver.svg	19.8 kB	Su	fem-run-solver.svg
fem-shell-thickness.svg	11.5 kB	Sat	fem-shell-thickness.svg
fem-solver.svg	5.5 kB	Fri	fem-solver.svg
fem-sphere.svg	4.4 kB	Su	fem-sphere.svg
fem-static-analysis.svg	65.1 kB	Su	fem-static-analysis.svg
fem-thermomechanical-analysis.svg	482.7 kB	Su	fem-thermomechanical-analysis.svg
fem-warp.png	999 B	Su	fem-warp.png
preferences-fem.svg	98.2 kB	Sat	preferences-fem.svg
translations	4.1 kB	Su	translations
Fem.qrc	3.6 kB	Su	**Fem.qrc**
UpdateResources.bat	57 B	Su	UpdateResources.bat
AbaqusHighlighter.cpp	6.6 kB	Sat	AbaqusHighlighter.cpp
AbaqusHighlighter.h	2.1 kB	Sat	AbaqusHighlighter.h
ActiveAnalysisObserver.cpp	3.5 kB	Sat	ActiveAnalysisObserver.cpp
ActiveAnalysisObserver.h	2.6 kB	Sat	ActiveAnalysisObserver.h
AppFemGui.cpp	6.6 kB	Su	*AppFemGui.cpp*
AppFemGuiPy.cpp	6.4 kB	Su	AppFemGuiPy.cpp
CMakeLists.txt	9.1 kB	Su	CMakeLists.txt
Command.cpp	53.5 kB	Su	**Command.cpp**
DlgSettingsFem.ui	14.6 kB	Su	DlgSettingsFem.ui
DlgSettingsFemCcx.ui	23.2 kB	Su	DlgSettingsFemCcx.ui
DlgSettingsFemCcxImp.cpp	4.5 kB	Su	DlgSettingsFemCcxImp.cpp

11.3.7 git setup for FreeCAD

- line encoding

For upstream master, line endings is `\r\n`.

```
git config --global core.autocrlf input
# Configure Git on OS X or Linux to properly handle line endings

git config --global core.autocrlf true
# Configure Git on Windows to properly handle line endings
```

- indentation by spaces
- removing Trailing space, especially for python source code
- backup files

After rebase, lots of `*.orig` files left in source folder. The git mergetool that produces these files, you can disable them with this command:

```
git config --global mergetool.keepBackup false
```

11.3.8 useful tips for git users

- Show history in graph or GUI

Gitg is a clone of Gitk and GitX for GNOME (it also works on KDE etc.) which shows a pretty colored graph. For textual output you can try: `git log --graph --abbrev-commit --decorate --date=relative --all` OR `git log --graph --oneline --decorate --date=relative --all` or https://git.wiki.kernel.org/index.php/Aliases#Use_graphviz_for_display is a graphviz alias for drawing the DAG graph.I personally use `gitx, gitk --all` and github.

- List only changes in one Mod folder

  ```
  git difftool [<options>] [<commit> [<commit>]] [--] [<path>]
  -g/--gui
  -d / --dir-diff
  git diff master..yourbranch path/to/folder
  git diff tag1 tag2 -- some/file/name
  ```

  Copy the modified files to a temporary location and perform a directory diff on them. This mode never prompts before launching the diff tool.

like `mergetool` GUI tool could be used to assist review diff

- Clone only one branch from other for testing

```
git clone -b foamsolver --single-branch https://github.com/qingfengxia/FreeCAD.git
```

- Undo your mis-conduct in git

How to undo (almost) anything with Git

```
git rm <stagedfile>
git checkout <changed but not staged file>
```

Always do a folder zip backup ,if you are new to git

- Consolidate/squash several commits into one clean commit before pull request

What I did: merge in my feature branch with the help of GUI

```
git pull --rebase upstream master
git checkout feature_branc
git rebase -i master
git mergetool -t meld
```

During rebase, there is a chance to squash commits.

11.3.9 Testing feature or bugfix

After test, git commit it and even git push to your repo. make a copy of the changed folder of the merged-with-upstream feature branch.

`git checkout master` and copy the folder back.

`git status` will show all the changed files in feature branch.

`git checkout -b feature_branch_clean` will make a patch/diff of all feature change wihte upstream master. git commit it after testing

`git push origin feature_branch_clean` and make a pull request online

Testing by macro or scripting

I taught myself a painful lession by attempting modifying many file before testing. Finally, I start again to refactoring single file and pass the test.

Unit test would be recommended, feeding predefined data input to automate the testing.

GUI debugging is time-consuming. FreeCAD has the macro-recording function, which can be used to save time on GUI testing by playing back macro.

FreeCAD is still under heavy development, testers are welcomed in every modules.

11.3.10 Procedure for user without a online forked repo (not tested ,not recommended)

As you don't have push (write) access to an upstream master repository, then you can pull commits from that repository into your own fork.

- Open Terminal (for Mac and Linux users) or the command prompt (for Windows users).
- Change the current working directory to your local project.
- Check out the branch you wish to merge to, usually, you will merge into master
- `git checkout master`
- Pull the desired branch from the upstream repository. `git pull upstream master`, This method will retain the commit history without modification.
- `git pull https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git BRANCH_NAME`
- If there are conflicts, resolve them. For more information, see "Resolving a merge conflict from the command line".
- Commit the merge.
- Review the changes and ensure they are satisfactory.
- Push the merge to your GitHub repository.
- git push origin master

11.3.11 Pull request and check feedback

It is recommended to submit small, atomic, manageable pull request to master, definitely after a full test.

After you push your commit to your fork/branch, you can *compare* your code with master. It is worth of coding style checking. For python code, using `flake`, `PEP8` etc. , `cppcheck` for C++ code.

Follow the standard github pull request routine, plus create a new post to describe the pull request, and wait for core developers/collobrators to merge.

11.3.12 example of pull request for bugfix

Spot out the bug: naming bug in Fem module: `StanardHypotheses` should be `StandardHypotheses`

1. find the bug and plan for bugfix

Assuming, current folder is Fem `/opt/FreeCAD/src/Mod/Fem`, find a string in all files in a folder, including subfolders: `grep -R 'StanardHypotheses' ./` output:

```
./App/FemMesh.cpp:void FemMesh::setStanardHypotheses()
./App/FemMesh.h:    void setStanardHypotheses();
./App/FemMeshPyImp.cpp:PyObject* FemMeshPy::setStanardHypotheses(PyObject *args)
./App/FemMeshPyImp.cpp:        getFemMeshPtr()->setStanardHypotheses();
./App/FemMeshPy.xml:        <Methode Name="setStanardHypotheses">
```

If not, then use find, try this: `find ./ -type f -exec grep -H 'yourstring' {} +`

2. make the patch and test locally

pull from the most updated upstream master, then make a new branch and checkout this branch `git checkout renamingFem`

replace a string in all files in a folder, including subfolders

```
grep -rl StanardHypotheses ./ | xargs sed -i 's/StanardHypotheses/StandardHypotheses/g'
```

check the result of replacement: There should be no output: `grep -R 'StanardHypotheses' ./` Then again: `grep -R 'StandardHypotheses' ./`, should match the file and lines number found in step 1

```
git add ./App/FemMesh.cpp
git add ./App/FemMesh.h
git add ./App/FemMeshPyImp.cpp
git add ./App/FemMeshPy.xml
```

`git commit -m "correct spelling StanardHypotheses to StandardHypotheses"`

Compile the source, make sure it can compile and function as expected. This function is not used in other module, so there is no need for function test.

3. submit pull request to upstream master

the push target is not official master, but developered github repo, see `git remote -v git push origin renamingFem`

On your project page of the github website, select this fork and *create pull request* to official master.A good description of bug and bugfix will make this pull request easier to be approciated.

Do it as quick as as possible, or this pull request will not be automatically merge with official master.

==

11.4 Code review

11.4.1 develop in developer's fork and test before PR

11.4.2 Travis-ci auto compiling for pull request

After a pull request on github, it will be compiled and unit test will be conducted automatically at 3 major platforms, win, macos and ubuntu.

`Continuous-integration/appveyor/pr - Waiting for AppVeyor build to complete`

`Required continuous-integration/travis-ci/pr - The Travis CI build is in progress`

The setup of Travis-ci for github is very straight-forward, see official manual https://docs.travis-ci.com/user/getting-started/

```
1. Sign in to Travis CI with your GitHub account, accepting the GitHub access permissions confirmation.

Once you're signed in, and we've synchronized your repositories from GitHub, go to your profile page and enabl

    Note: You can only enable Travis CI builds for repositories you have admin access to.

2. Add a .travis.yml file to your repository to tell Travis CI what to build:

3. Add the .travis.yml file to git, commit and push, to trigger a Travis CI build:

    Travis only runs a build on the commits you push after adding the repository to Travis.

4. Check the build status page to see if your build passes or fails.
```

However, for module developers, the free compiling service may not be so usful. The compiling of FreeCAD code base may be too long to finish within free time limit.

11.4.3 code review tool and process

code review tool and process

Phabricator looks really promising - there are tons of options, so I'll be posting things that might be useful for us.

1. We should use "review" workflow (commit is reviewed before is included in the master branch). More here [1]
2. Phabricator can host git repository, can tract remote repo (that's what is configured right now) and can use mirrors. What we need is not clear for me yet.
3. We'd need at least virtual server to set it up - there are some tweaks in mysql/php required, so a normal cloud hosting might not be enough.
4. The system right now runs on raspberry pi model 2 B (4 core, 1GB, 100Mb ethernet), and is connected over my home broadband (240/20Mb), so any virtual server should be more than enough to run it.
5. Configuration of the system is "I'll guide you by the hand" (Please set variable X = Y in file /etc/myslq/whatever) or GUI driven. It's easy.
6. It's handy to have mail server configured (postfix), for notifications/password reset.

7. Setting up dashboard (page that users see as the main page) - it's gui driven and very easy.
8. There are github integration options - I did not explore them yet.

[1] https://secure.phabricator.com/book/pha ... _vs_audit/

Chapter 12

FreeCAD coding style

12.0.1 text encoding and spaces

- source code encoding "utf8": defaultl for most IDE
- indentation, never use TAB, but 4 spaces
- end of line (EOL): using windows style `\r\n` set in git and your IDE tool,

Not tested shell script, only if all EOL is `\n`:

```
# remove carriage return
sed -i 's/\r//' CRLF.txt

# add carriage return
sed -i 's/$/\r/' LF.txt
```

It is worth of print before substite: "

- remove trailing whitespace: search by: `find ./*.cpp -type f -exec egrep -l " +$" {} \;` search before you replace all, some file needs trailing whitespaces, print `sed -n /[[:blank:]]*$/p file.cpp` substitute with: `sed 's/[[:blank:]]*$//' file.cpp`
- no trailing spaces at end coding: search by command"
- limitation of max char in one line: make it easy to read without scrollbar, 80-90 is recommended.
- doxygen in source documentation

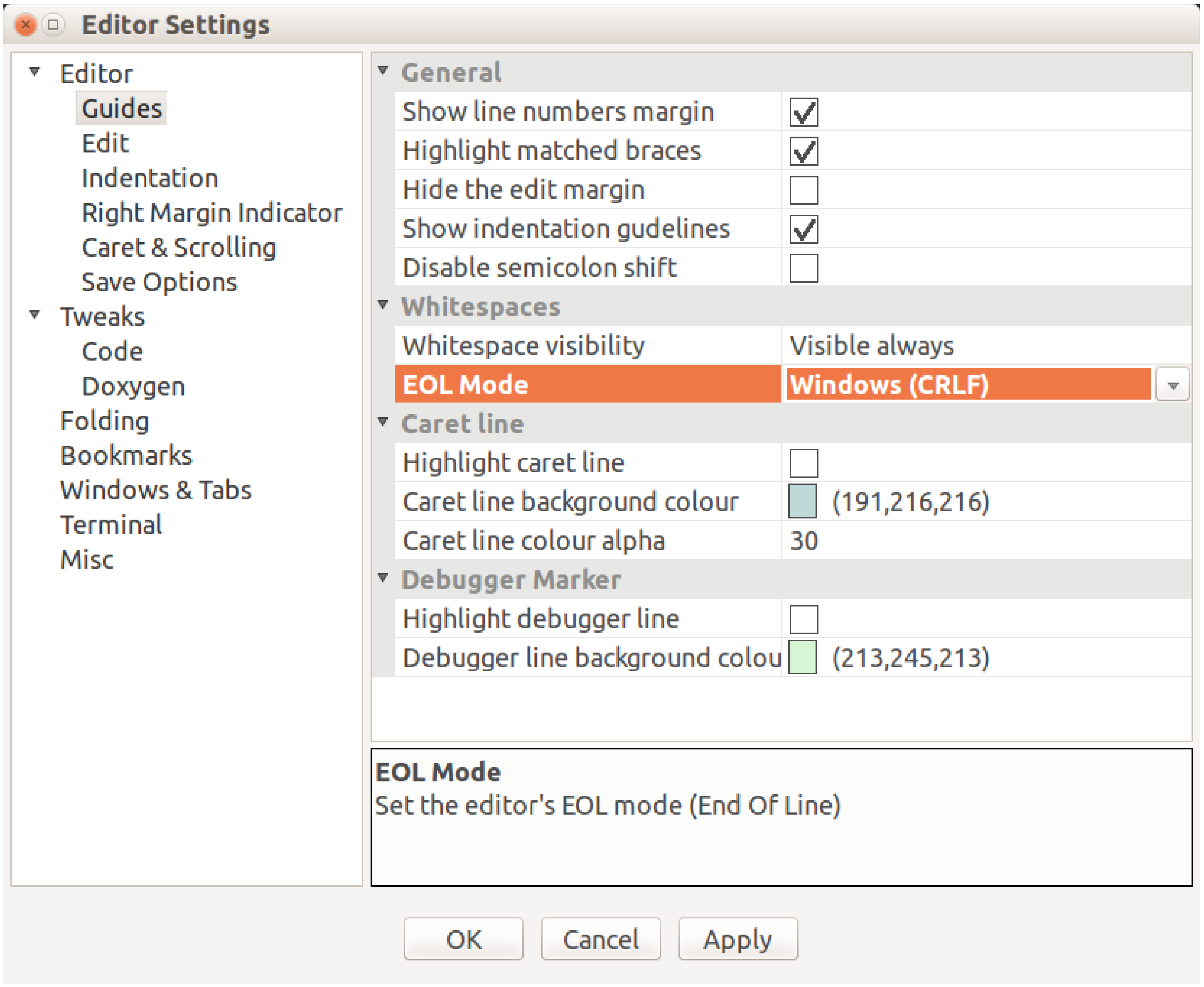
12.1 C++ coding style

12.1.1 close to Qt style

Generally, the C++ coding style is similar with Qt http://qt-project.org/wiki/Qt_Coding_Style

for example

```
StdCmdExport::StdCmdExport()
    : Command("Std_Export")
{
     // setting the
     sGroup        = QT_TR_NOOP("File");
     sMenuText     = QT_TR_NOOP("&Export...");
     sToolTipText  = QT_TR_NOOP("Export an object in the active document");
     sWhatsThis    = "Std_Export";
     sStatusTip    = QT_TR_NOOP("Export an object in the active document");
     //sPixmap       = "Open";
     sAccel        = "Ctrl+E";
```



General	
Show line numbers margin	✓
Highlight matched braces	✓
Hide the edit margin	☐
Show indentation gudelines	✓
Disable semicolon shift	☐
Whitespaces	
Whitespace visibility	Visible always
EOL Mode	Windows (CRLF)
Caret line	
Highlight caret line	☐
Caret line background colour	■ (191,216,216)
Caret line colour alpha	30
Debugger Marker	
Highlight debugger line	☐
Debugger line background colou	🟩 (213,245,213)

EOL Mode
Set the editor's EOL mode (End Of Line)

Figure 12.1: codelite end of line and space visuability

```
    eType         = 0;
}
```

type prefix for function parameter is not as useful as for member Variable,

- i: integer
- s: char const*, std::string
- p for pointer (and pp for pointer to pointer)
- pc: pointer of C++ class
- py: pointer of Python object
- _privateMember

for example: `App::DocumentObject *pcFeat`

It is more Coin3D style,except "So" namspace suffix is not used. In 2003, C++ compilers are not so powerful and standardised to support even template and namespace in a cross-platform way. visual c++ was really bad to support C++ standard for some time.

- Namespace is enforced for each module, using "Export"
- class name (CamelClass) , Acronyms are camel-cased like 'XmlWriter'
- private members:
- member function name (begins with lowerCase).
- no tab but 4 spaces indentation

12.1.2 Fifference from Qt style

- getPropertyName() is used in FreeCAD, while propertyName() is used in Qt, more example on getter names

  ```
  Gui::Application::Instance
  Gui::MainWindow::getInstance();
  Gui::getMainWindow();
  ```

- function parameter has the pattern "a single char for type"+"meaningful name" > common type char: s->string; i->int; h->Base::Reference/object handle; e->enum; f->float/double; p->pointer;

- c++ STL and boost lib is used, but higher level Qt style API provided for user

---

12.2 Python coding style

12.2.1 Disccusion on Python coding standard

Forum Disccusion on Python coding standard

if API will be exposed to other user, QtSide coding style should be adapted as possible

python standard coding style could be used internally.

property name start with upppercase, e.g. src/Mod/TemplatePyMod/DocumentObject.py

PythonCommand class name exportable class should follow NamingConvention doSomthing()

```
Command<SpecificName>
ViewProvider<>
<>TaskPanel
_PrivateClassName
```

12.2.2 style checker

PyCXX (Py::Object) should be used as possible, it may give better python2.x and python 3.x compatibility over the raw C API in `<Python.h>`

return `PyObject*` and `Py::Object` has different impact pep8 and pyflake to check coding style: sudo apt-get install python3-flake8 flake8 `flake8 --ignore E265,E402,E501,E266 yourfile.py`

https://github.com/google/yapf

Python IDE would suggest confliction with flake8 and avoid trailing spaces in c++ IDE

12.3 Inconsistent of naming

src/Mod/Part/JoinFeatures.py `obj.Mode = ['bypass','Connect','Embed','Cutout'] // bypass should be upcase ?`

src/Mod/Part/App/TopoShape.h `static void convertTogpTrsf(const Base::Matrix4D& mtrx, gp_Trsf& trsf);`

this should be a typo

Chapter 13

Cmake cheat sheet

13.1 quick start

1. first example, see more details at https://cmake.org/cmake-tutorial/

```
cmake_minimum_required (VERSION 2.8.0)#cmake grammar evolves,
project (testcpp) # define the output executive name

add_executable (testcpp testcpp.cpp) #define source to object relation
```

2. in source build cmdline:
   `cmake <path to toplevel cmakelists.txt> -G <generator> && make`

out of source build is recommended to keep source tree clean:

```
cd dir_of_source_with_toplevel_cmakelists.txt && mkdir build
cd build && camke .. &&make
make install
```

using `nmake` on windows visual studio build tool, as cmake generate platform specific makefile

3. GUI tool `cmake-gui <path to toplevel cmakelists.txt>` build option can be turn on and off in GUI

   ```
   option(<option_variable> "help string describing option"
      [initial value])
   ```

4. FindGTK module, work like `pkg-config` to get include and lib path

```
GTK_INCLUDE_DIR   - Directories to include to use GTK
GTK_LIBRARIES     - Files to link against to use GTK
GTK_FOUND         - GTK was found
GTK_GL_FOUND      - GTK's GL features were found
```

for more details see https://cmake.org/Wiki/CMake

13.2 cmake language syntax

cmake is a script language, similar with shell to some extent. see cmake wiki

- only one kind of variable, i.e. string,
  > but it will be interpreted as bool in `IF (var)`
  or integer `IF (var GREATER 10) ... ENDIF`
- variable substitution `${VAR}`, SET and UNSET, `IF(DEFINED x)`
- only one kind of data structure: LIST, string sperated by semicolon
  `SET(var a b c)` is equal to `SET(var "a;b;c")`
- quotation mark to reserve sapce “hello world”

```
SET( x a b c   ) # stores a list "a;b;c" in x      (without quotes)
SET( y "a b c" ) # stores a string "a b c" in y      (without quotes)
MESSAGE( a b c ) # prints "abc"   to stdout, automatically concat (without quotes)
MESSAGE( ${x} )  # prints "abc"   to stdout (without quotes)
MESSAGE("${x}")  # prints "a;b;c" to stdout (without quotes)
MESSAGE( ${y} )  # prints "a b c" to stdout (without quotes)
MESSAGE("${y}")  # prints "a b c" to stdout (without quotes)
```

- comment by #, escape by , line continuation by / > but in path string: "http://///www.example.com"

  but it is different from bash shell

  newline to split list element

- cmake variable and command is case insensitive
  > but UPPER case for command is recommended
  and variable content like filename is case sensitive

- ""(empty string), "NO; N; FALSE; -NOTFOUND; OFF;0;" or lower case is regarded false in IF()

- only integer algorithm `MATH(EXPR outputvar expr)`

```
SET( expression 4 LESS 10 ) # ${expression} is now "4;LESS;10"
IF( ${expression} )         # expands to IF( 4;LESS;10 )
  MESSAGE( "CMake believes that 4 is less than 10." )
ENDIF( ${expression} )
```

  bash shell math: `$((math_expr))`

- generate `config.h` from `config.h.cmake` template
  > by replacing "@var" with var defined in camkelists.txt `MESSAGE( ${x}${y} ) # displays "31"`

- function is different from macro whose variables have global scope

13.3 important commands and variables

see more details on variable at https://cmake.org/Wiki/CMake_Useful_Variables

13.3.1 1.commands

- $ENV{HOME} #access to environment variable
- EXECUTE_PROCESS(cmd var_to_hold_result)
- INSTALL(TARGETS|FILES list_of_f dest)
- CONFIGURE_FILE() #generate config.h from config template input file
- INCLUDE_DIRECTORY() #subdir with cmakelists.txt
- ADD_LIBRARY(), ADD_EXECUTABLE(), add_definitions, add_dependencies
- INCLUDE(extra_cmake_file) #as include in c++ for extra functions

13.3.2 2. location variables

- SET(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/MyCMakeScripts) > where FIND_PACKAGE(FooBar) can find `findFooBar.cmake`

- CMAKE_SOURCE_DIR # top source dir containing the top-level CMakeLists.txt,

- CMAKE_BINARY_DIR # top folder to build binary object

- CMAKE_CURRENT_LIST_FILE # full path of the current processed cmakelists.txt

- CMAKE_INCLUDE_PATH # path to search for header file e.g. `/usr/include`

- CMAKE_INSTALL_PREFIX #just as DESTDIR in `make DESTDIR=/home/john install`

13.3.3 3. compiler control variables

- CMAKE_CXX_COMPILER
- CMAKE_BUILD_TYPE # NONE, DEBUG, RELEASE, etc
- CMAKE___FLAGS # for build_type NONE
- CMAKE_LINK_LIBRARY_CFLAGS
- enable C++ standard support in a portable way

  ```
  set(CMAKE_CXX_STANDARD 11) #98, 14, 17
  set(CMAKE_CXX_STANDARD_REQUIRED ON)
  ```

13.3.4 4. predefined platform and compiler detection variables

platform like: WIN32, APPLE, UNIX (linux is UNIX but not APPLE), MYSYS, CYGWIN compilers: MSVC, MINGW, iNTEL; CMAKE_COMPILER_IS_GNUCXX, CLANG

13.3.5 5. command line help

```
cmake --variable-list
cmake --command-list
cmake --module-list
cmake -h #show all generator
```

https://cmake.org/Wiki/CMake_Useful_Variables

13.4 Why cmake?

cmake is a new language, why should we learn a new language instead of json or python (as in scons)?

1. cmake inherits some conception of traditional unix makefile
2. multiple programming lang support and cross-platform with generators for majourity of IDE tools
3. plenty of official modules like “FindXXX.cmake”
4. speedup by var cache `SET(var value CACHE)`
5. CPACK for binary package and windows NSIS installer generation
6. Generating Dependency Graphs with CMake, `cmake --graphviz=test.dot .`
7. tool to generate cmakelists.txt from scratch or convert from existent build system

13.5 what I do not like

- cmake is not a precise or clear script lang
- confusing string quotation and substitution
- un-intuitive function style programming to c/python programmers
- messed up of variable_content and variable
- mixed up argument list and keyword in function call, not sep by comma > relies heavily on this function to parse function/command input arg `CMAKE_PARSE_ARGUMENTS(<prefix> <options> <one_value_keywords> <multi_value_keywords> args...)`

Chapter 14

Learning OpenInventor/Coin3D

Coin3D is an open source implementation for OpenInventor spec/API, released in a less constrainted license, LGPL.

Usful links to learn OpenInventor programming: http://webee.technion.ac.il/~cgcourse/InventorMentor/The%20Inventor%20Mentor.pdf Coin3D Online Document

14.0.1 Important classes in OpenInventor/Coin3D

`SoPath, SoNode, SoEngine` are three main categories of Object in Coin3D. Classes are organised into modules, see http://developer90.openinventor.com/APIS/RefManCpp/main.html

Description from this online documentation is extracted for key classes. See the brief description for classes: http://coin3d.bitbucket.org/Coin/annotated.html;

** Basic objects **

- SbXXX: Basic types like SbVec3f, SbMatrix, SbColor, SbString, SbTime; Containers like SbDict, SbList; geometrical representation of basic shape like SbSphere; SbTypeInfo
- SoBase: ancester for most Coin3D ojbects, similar with *QObject*, FreeCAD's `Base::BaseClass`

  Top-level superclass for a number of class-hierarchies. SoBase provides the basic interfaces and methods for doing reference counting, type identification and import/export. All classes in Coin3D which uses these mechanisms are descendent from this class

```
ref() unref() getName()
virtual SoType  getTypeId (void) const =0
notify (SoNotList *nl) //observer pattern, notify Auditor
addAuditor (void *const auditor, const SoNotRec::Type type)
```

Qobject is the base object for all derived Qt objects, offering event, container, property, type support.

Example of inheritance chains:

Coin3D: `SoBase->SoFieldContainer->SoNode->SoGroup->SoShape` FreeCAD: `BaseClass->App::PropertyContainer->App::Docu`

- SoType: Inventor provides runtime type-checking through the `SoType` class. `node->getTypeId().getName();` like `Base::TypeClass` in FreeCAD

  Basis for the run-time type system in Coin3D. Many of the classes in the Coin3D library must have their type information registered before any instances are created (including, but not limited to: engines, nodes, fields, actions, nodekits and manipulators). The use of `SoType` to store this information provides lots of various functionality for working with class hierarchies, comparing class types, instantiating objects from classnames, etc etc
- SoField: Top-level abstract base class for fields serializable, similar with `App::Property` in FreeCAD

  Fields is the mechanism used throughout Coin for encapsulating basic data types to detect changes made to them, and to provide conversion, import and export facilities. *SoSFXXX*: Single Field with Base type wrapped (App::Property); *SoMFXXX*: Multiple Field (array of field). E.g. SoSFBool class is a container for an SbBool value.

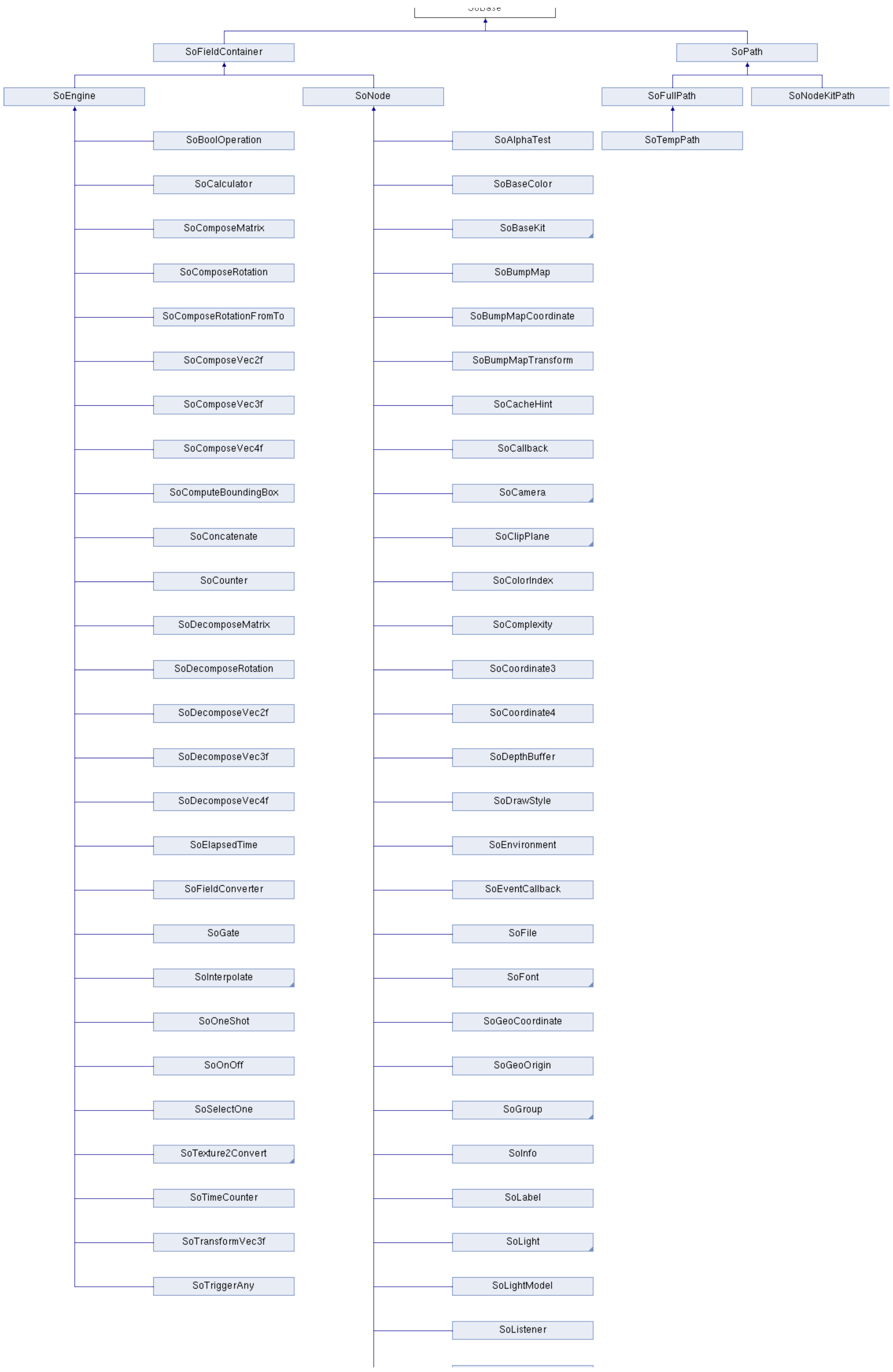


Figure 14.1: Inheritance chain of Coin3D

- SoFieldContainer: serializaton(`App::PropertyContainer` in FreeCAD) function is built into `SoNode`
- SoBaseList Container for pointers to SoBase derived objects.

  The additional capability of the SoBaseList class over its parent class, SbPList, is to automatically handle referencing and dereferencing of items as they are added or removed from the lists

** Scene organisation **

- SoDB: This class collects various methods for initializing, setting and accessing common global data from the Coin library

  Similar with `App::Document` in FreeCAD import and export into file. Directed Acyclic Graph is used for better performance, SoNodes are organised into database, serialization into text file *.iv .

  "The foundation concept in Open Inventor is the"scene database" which defines the objects to be used in an application. When using Open Inventor, a programmer creates, edits, and composes these objects into hierarchical 3D scene graphs (i.e., database). " Quoted from Open Inventor reference.

- SoNode: similar with `App::DocumentObject` in FreeCAD, has flags like ignore, override

  Base class for nodes used in scene graphs. Coin is a retained mode 3D visualization library (built on top of the immediate mode OpenGL library). "Retained mode" means that instead of passing commands to draw graphics primitives directly to the renderer, you build up data structures which are rendered by the library on demand

- SoGroup: similar with `App::DocumentObjectGroup` in FreeCAD

  An `SoSwitch` node is exactly like an SoGroup except that it visits only one of its children. `SoShape` is derived from `SoGroup` Shared Instancing: share the SoShape, but separate SoTransform, ref counting

- SoSeperator: State-preserving group node (derived from `SoGroup`), conprising SoColor, SoMaterial, SoTexture, SoShape, etc.

  Subgraphs parented by SoSeparator nodes will not affect the previous state, as they push and pop the traversal state before and after traversal of its children. Order (topdown, left to right) in SoDB (scene graph) is important to determine rendering, see example in http://developer.openinventor.com/content/34-creating-groups. Scale node is only added to first Hydrogen SoGroup, but this scale applied to the second Hydrogen SoGroup. To isolate the effects of nodes in a group, use an SoSeparator node, which is a subclass of SoGroup . Before traversing its children, an SoSeparator saves the current traversal state. When it has finished traversing its children, the SoSeparator restores the previous traversal state. Nodes within an SoSeparator thus do not affect anything above or to the right in the graph.

- SoPath: Container class for traversal path for nodes in scene database, see also `SoFullPath, SoNodeKitPath`. It is derived from `SoBase`, not `SoFieldContainer`, it is different from `App::PropertyLink` in FreeCAD.

  "SoPath objects contain a list of SoNode pointers and a list of child indices. Indices are necessary to disambiguate situations where a node uses the same node as a child multiple times. Similarly, UUID and getUniqueName() in FreeCAD make the unique reference to Document Objects."

- SoBaseKit: base class for all NodeKit (not a SoGroup) which create groups of scene graph nodee. Parts are added as hidden children, accessible only by the methods of SoBaseKit and its derived classes.
- SoSeparatorKit: A nodekit that is used for creating nodekit hierarchies. SoSeparatorKit contains a transform part, a childList part, and a few others like pickStyle , appearance in its catalog.

** Scene rendering **

- SoAnnotation: (Derived from SoSeparator) node draws all its child geometry on top of other geometry.

  This group-type node uses delayed rendering in combination with Z-buffer disabling to let its children transparently render their geometry on top of the other geometry in the scene.

- SoShape: SoCube/SoCone/SoCynlinder/SoSphere/SoText/SoImageSoNurbsCurve/SoNurbsSurface/SoImage: (`App::GeoFeature` in FreeCAD??)

  For rendering basic shapes.Insert a shape into the scenegraph and render with the current material, texture and drawstyle settings (if any, otherwise the default settings are used)

- SoDetail: Superclass for all classes (SoCubeDetail. . .) storing detailed information about particular shapes.

Detail information about shapes is used in relation to picking actions in Coin. They typically contain the relevant information about what particular part of the shape a pick ray intersected with

** misc objects **

- SoEngine: SoEngine (derived from SoFieldContainer, as a sibling of SoNode) is the base class for Coin/Inventor engines.

  Engines enables the application programmers to make complex connections between fields, for example, animation.

- SoVRMLXXX: VRML file import and export
- SoAudioDevice: 3D sound
- SoSensor: for scene manipulation
- SoCamera: belongs only to scene
- SoLight: belongs only to scene
- SoEnvironment: gloable settings
- ScXml: Namespace for static ScXML-related functions
- SoElement: base class for classes used internally for storing information in Open Inventor's traversal state list.
- SoSelection: Manages a list of selected nodes, Derived from SoSeparator.

  Inserting an SoSelection node in your scene graph enables you to let the user "pick" with the left mousebutton to select/deselect objects below the SoSelection node

- SoSFEnum/SoMFEnum: single or multiple Enumeration fields

** Action, event and callback **

- SoAction: SoCallback(object oriented)

  Applying actions is the basic mechanism in Coin for executing various operations on scene graphs or paths within scene graphs, including search operations, rendering, interaction through picking, etc

- SoEvent: Base class for keyboard/mouse/motion3d event
- SoEventCallback: nodes in the scenegraph for catching user interaction events with the scenegraph's render canvas
- SoCallback: Node type which provides a means of setting callback hooks in the scene graph.

  By inserting SoCallback nodes in a scene graph, the application programmer can set up functions to be executed at certain points in the traversal - SoCallbackAction: Invokes callbacks at specific nodes.This action has mechanisms for tracking traversal position and traversal state.

  In combination with the ability to pass geometry primitives to callback actions set by the user, this does for instance make it rather straightforward to extract the geometry of a scene graph

- SoCallbackList The SoCallbackList is a container for callback function pointers, providing a method for triggering the callback functions

see http://developer.openinventor.com/content/chapter-10-handling-events-and-selection

14.0.2 Window System integration

Previous (deprecated) windwos system integration lib:

- SoWin: for win32 windows platform
- SoXt: for XWindows for *nix system
- SoQt: integrating with Qt window system

Quater: the most updated bind with Qt Quarter is superior over SoQt providing OpenGL widget viewer. Release 1.0.0 is the first major release. Quarter 1.0 is only usable with Coin-3.x and Qt-4.x.

Quarter is a light-weight glue library that provides seamless integration between Systems in Motions's Coin high-level 3D visualization library and Trolltech's Qt 2D user interface library, to replace SoQt. The functionality in Quarter revolves around QuarterWidget, a subclass of `QGLWidget`. This widget provides functionality for rendering of Coin scenegraphs and translation of `QEvents` into SoEvents. Using this widget is as easy as using any other `QWidget`.

Quarter / include / Quarter / QuarterWidget.h

For developers targeting multi-platform - ‘Quarter’ provides a seamless integration with the Qt framework. https://en.wikipedia.org/wiki/Coin3D

http://doc.coin3d.org/Quarter/