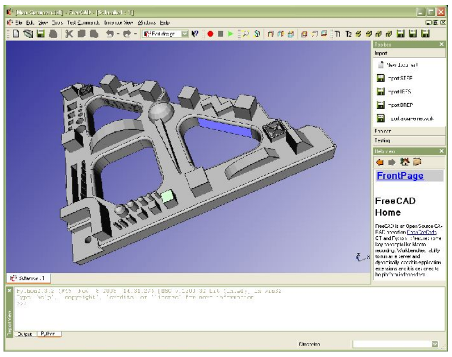
FreeCAD
a free extensible CAx system

Jürgen Riegel

juergen.riegel@web.de
2002

Index

1 Introduction



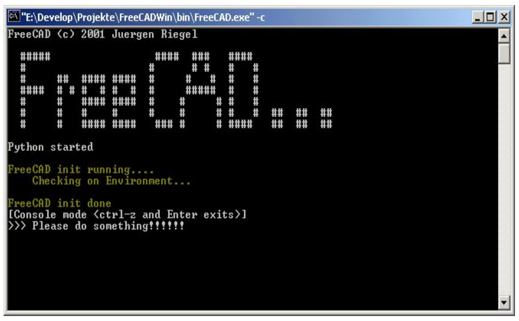
With the release of Open CasCade in 2000 the opportunity was given to use this class librarie to plan and implement the first fully featured free 3D CAD system for mechanical engineering. For me was the point my experience on the CasCade Kernel and the my work as a profesional programmer where I work more with Powerpoint rather using the compiler.

Without experience in professional work one would have great trouble to get into the concepts and algorithms of the CasCade Kernel as a whole. So the plan is to provide a framework which utilize the important packages of the CasCade Kernel (topology, OCAF, I/O) and make it easy for people provide functionality at here own level of skills. Anyway, to program a application with the size and the function richness of a CAD package on your own is perfectly impossible. So my hope is that this project attract developers all over the world to contribute.

Also for me it’s a interesting thing, since I’m involved in programming a CasCade application (in the field of reverse engineering) at DaimlerChrysler, to use the lessons I’ve learned and introduce them in a completely new design. This work is done in my spare time and in no way related to DaimlerChrysler or the Research department of DC I’m currently working.

Jürgen Riegel 2000

2Tutorial



```
FreeCAD (c) 2001 Juergen Riegel

  #####                 ####  ###   ####
  #                    #      # #   #   #
  #     ##  #### ####  #     #   #  #   #
  ####  # # #  # #  #  #     #####  #   #
  #     #   #### ####  #    #     # #   #
  #     #   #    #     #    #     # #   #  ##  ##  ##
  #     #   #### ####   ### #     # ####   ##  ##  ##

Python started

FreeCAD init running....
    Checking on Environment...

FreeCAD init done
[Console mode <ctrl-z and Enter exits>]
>>> Please do something!!!!!!
```

The first point will be a tutorial to enable the reader to use FreeCAD on user level.

2.1General

2.1.1FreeCAD with the GUI

Most user will use FreeCAD with the Graphical User Interface (Gui).

2.1.2FreeCAD in Command mode

Starting

For more sophesticated work and batch jobs or server work you can start FreeCAD as a console application. Just type:

```
FreeCAD -c
```

and the Gui whont appear. Instad will FreeCAD promt you for commands after the initialization has finished. There is also a special executable for server and batch jobs:

```
FreeCADCmd
```

This has no grfical user interface et all and its there fore smaller and need less resources.

To start FreeCAD non interactive give a batch file as a parameter:

```
FreeCAD -cf DoALongBatchJob.FCScript
```

This will cause FreeCAD to start imidatly the given file and terminate when the script is finished. You can use batch files i.g. For:

- automating often done jobs without user interaction like updating large Assambly structures after changes or converting thousends of files in an other format
- starting scripts which open a server and process transactions. Python allows to opens TCP/IP ports and process requests from other systems. Its also possible to instanciate a whole webservice.
- Runing macros recordet with the user interface
- and so on .....

Interactive Command mode

After starting FreeCAD in with the -c option you will see somthing like this:

```
FreeCAD (c) 2001 Juergen Riegel (GPL,LGPL)

  #####                 ####  ###   ####
  #                    #      # #   #   #
  #     ##  #### ####  #     #   #  #   #
  ####  # # #  # #  #  #     #####  #   #
  #     #   #### ####  #    #     # #   #
  #     #   #    #     #    #     # #   #  ##  ##
  #     #   #### ####   ### #     # ####   ##  ##

FreeCAD startup running....
    Checking on Environment...
FreeCAD startup done
Create Application
FreeCAD init running....
   Using ..\src\Mod as module path!
   Searching modules...

FreeCAD init done
[Console mode <ctrl-z and Enter exits>]
>>>_
```

The last line is a python command prompt. As FreeCAD includes a complet python interpreter you can use all python build in features and modules.

2.2Part

Here the Part Workbench, which includes the most CAD functions.

2.3Mesh

The Mesh Module is dedicated to triangle mesh creation and functions.

2.3.1General

2.3.2Scripting

Besides the way of using meshes with the document and the GUI, it's also possible to use the mesh data and algorithems on script level. Thats opens the way to do mesh calculations in macros or just on the command line (FreeCADCmd.exe).

Basics

First of all you have to load the FreeCAD Mesh moule:

```
    #import the Mesh functions
    import Mesh
    # shows you the methodes of the Mesh module
    dir(Mesh)
```

After that you can use all the functions in the mesh module

Import / Export

You can read / write following formates:

- STL (stero lithographie format)
- BMS (nativ binary, the evicent way to write read meshes)

Example:

```
    m = Mesh.read("Something.stl")
    # Do something realy exiting with the mesh
    m.write("Another.stl")
```

Algorithems

Python works only with references. That means:

```
    m2 = m
```

creates no new mesh object. So some of the Algorithems creates new meshes as a result, other just impact the existing one.

You need to do:

```
    m2 = m.copy()
```

to get a new, copied, mesh object.

Creating meshes

Bisides loading on the creation of mesh is a way to make a mesh from scratch. The methode addFace() of the mesh object can do the job:

```
    # create a new empty mesh
    m = Mesh.newMesh()
    # build up box out of 12 facets
    m.addFacet(0.0,0.0,0.0, 0.0,0.0,1.0, 0.0,1.0,1.0)
    m.addFacet(0.0,0.0,0.0, 0.0,1.0,1.0, 0.0,1.0,0.0)
    m.addFacet(0.0,0.0,0.0, 1.0,0.0,0.0, 1.0,0.0,1.0)
    m.addFacet(0.0,0.0,0.0, 1.0,0.0,1.0, 0.0,0.0,1.0)
    m.addFacet(0.0,0.0,0.0, 0.0,1.0,0.0, 1.0,1.0,0.0)
    m.addFacet(0.0,0.0,0.0, 1.0,1.0,0.0, 1.0,0.0,0.0)
    m.addFacet(0.0,1.0,0.0, 0.0,1.0,1.0, 1.0,1.0,1.0)
    m.addFacet(0.0,1.0,0.0, 1.0,1.0,1.0, 1.0,1.0,0.0)
    m.addFacet(0.0,1.0,1.0, 0.0,0.0,1.0, 1.0,0.0,1.0)
    m.addFacet(0.0,1.0,1.0, 1.0,0.0,1.0, 1.0,1.0,1.0)
    m.addFacet(1.0,1.0,0.0, 1.0,1.0,1.0, 1.0,0.0,1.0)
    m.addFacet(1.0,1.0,0.0, 1.0,0.0,1.0, 1.0,0.0,0.0)
    # scale to a edge langth of 100
    m.scale(100.0)
```

This example shows how to build a cube out of 12 triangles. If the vertexs are the same the triangles get topological conected.

Offset

This function creats an offset Mesh allong the Vertex normals. That means

3 Specification

The Specification gives you an overview what we try to achieve. What kind of functions and features FreeCAD should have.

3.1 Overview

FreeCAD will be a general purpose 3D CAD modeller. The development will be completely as Open Source. As a modern 3D CAX modellers it will have a 2D component to extract design drawings from the 3D model, but 2D is not the focus, neither are animation and organic shapes (Maya, 3D StudioMAX, Cinema 4D). FreeCAD will aim directly to mechanical engineering, product design and related features (like CatiaV4 and V5, and SolidWorks). It will be a **Feature-Based parametric modeler**.

CAD modelers tend to become very large systems. They often incorporate thausend of functions whis can be split in areas of usage, like i.g.

There for a list of Features such a CAD modeller should have:

3.1.1 Workbenches

Modern applications nowadays often have to much functionality to show it to the user at once. Especially CAD modelers are known to have thausends of functions or more. Therefore its much better to package the functions in so called **Workbenches** show the user only the functions and controls he need for the actual design step. This keeps the menus an toolbars slim and easy to use. For the Feature-based modeling a natural grouping has evolved:

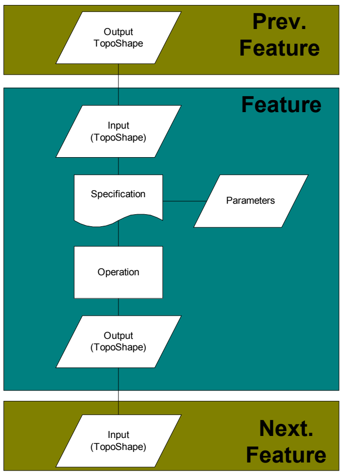
- **Sketcher**
  Constructing points, lines, arcs and free form curves on base of a plane (2D) with the ability to define constrains between them like dimensions, parallelism, coincidence, and so on. . .
- **Part Design**
  Functions for full parametric Solid Modelling like fuse, cut, chamfers, fillets, holes, slots and rips based on **Sketches**
- **Surface Design**
  Functions for Surface Modelling like extrudes, blends, trimming, lofts and filling surfaces based on **Sketches**
- **Assembly**
  Design Functions to assemble parts, designed in S**urface Design** or **Part Design**, together to Assemblies. Its also possible to define constrains between the Parts like axes, contact, and so on. Also functions to test inter-sections and movements (inverse cinematic)
- **2D Drawing Extraction**
  extract 2D drawings from the 3D **Part** or **Assembly** for printing and plotting
- **User defined Workbench**
  means the user can easily by macro recording and dialogs customis his own Workbench for special purpose

Later on additional Workbenches could be:

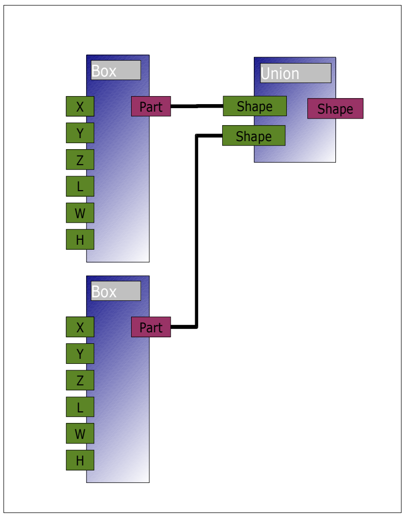
- **Rendering support**
  I/O support for RIB format for geometry and Shader, used by renderer like BMRT
- **Part Database**
  For storing and retrieving standard designs using a catalog functionality with previewing sharing in work groups.
- **FEM**
  Pre- and Postprocessor for meshing, defining constraints and viewing results. Transparent usage of a freely available FE simulation package (modal and stress analyses)
- **Special workbenches for:**
  Sheet metal design, moulding and casting tools
- And a whole lot more ….

3.1.2Feature-based parametric modeling

Since the begining of CAD (Computer Aided Design) the software and the key concepts of CAD have gone a long way. It started with a bunch of not acociatat lines on the screen and ends nowadays at the Feature-base parametric modeling.
Parametric modeling came up with the solid modelers. It allows to base a design on a set of parameters and force the recomputation of the geometry when the parameter change. This also lead to constrains which are introdused to the model, force a certain behaviour like parallelism. A constraint solver keep the constrains in touch with the geometry.
The finall step whas to group certain parameters and often used geometical pattern to a so called **Feature**. Which allow to automice common task and give a higher level representation.
A FreeCAD Feature basically works that way:



A good example for a feature is a hole. The input is the face of a solid part and the creation point, and the parameters are e.g. the diameter and the depth. With this Specification the Operation punch a hole in the solid part (Input) and generate a solid part with a hole (Output). If you change either the Input or the parameters the Operation get done again.
A Feature can be rather complex like patterns or complete ribs.



This Features can freely linked together to a model specification. Like the example on the right side, two Boxes are linked together with a Union Feature.
The whole FreeCAD document is a bunch of linked Features which have input parameters and output (shapes). One can see the docuement as a geometrycal function and the Features as the progamming steps.

3.1.3User interface

Besides the functional features the ease of use of a application is very important, there for the specification goes also for that and defines some cornerstones for the user interaction:

- **Workbenches**
  Its to much to show all the functions at once to the user. Its much better to packaging function needed for certain tasks together. That's a Workbench. The appearance of a Workbench consist of the Toolbar and menu layout as well as the visual mode of the project tree view. So the switching of the Workbench can nearly change the complete appearance of the application.
- **Guided selection**
  In CAD programs its very important to know what to select in a function. There should always be a guidance for the user and a clear response on wrong selection. This is first a clear dialog box layout which shows clearly what to select.
- **Enough online Help ;-)**
  That includes detailed descriptions on workbenches and functions as well as tutorials on best practice scenarios.

Also some important points for the usability of the application itself:

- **Familiar lockout**
  No fancy "skins" and experimental controls no animations, just a plain and simple MDI frame and all the menus at the familiar place
- **Customisation**
  There shut be a rich set of customisation mechanisms to al-low the user to change the standard behaviour the way he wish. That can be:
  - A easy to use macro recording facility o Implementation of parts of the application logic in a script language to allow the user to change it o Allow extensions implementation in a script language
- **Modularity**
  The program must be one big block. Function and Workbenches sould be loaded dynamically only when needed

3.2Licence

I know that the discussion on the "reight" licence for open source occupied a significant portion of internet bandwidth and so is here the why , in my opinion, FreeCAD should have this one.

I chooses the LGPL and the GPL for the project and I know the pro and cons about the LGPL and will give you some reasons for that decision.

FreeCAD is a mixture of a Library and an Application, so the GPL would be a little bit strong for that. It would prevent write commercial modules for FreeCAD because it would prevent linking with the FreeCAD base libs. You may ask why commercial modules at all? Therefore Linux is good example. Would Linux be so successful when the GNU C Library would be GPL and therefore prevent linking against non GPL Applications? And although I love the freedom of Linux but I want also use the very good NVIDIA 3D graphic driver. And I understand and accept the reasons of NVIDIA not give away here driver code. We all work for companies and need payment or at least foot. So for me a coexistence of open source and closed source software is not a bad thing, when it obeys the rules of the LGPL. I would like to see someone write a Catia import/export processor for FreeCAD and distribute it for free or for some money. I don't like to force him to give more away as he want to. That wouldn't be neither good for him, nor for FreeCAD.

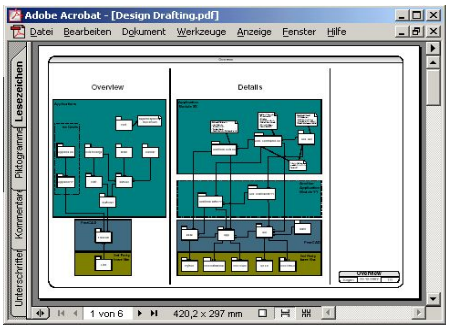
3.3Framework

3.3.1Overview

You can see FreeCAD as an Application but that's only the half of the truth. The executable itself has basically no modeling function. It is more or less only a basic framework to allow application modules to register and offer functions to the user. So the FreeCAD executable is best named as a "framework" for embedding the functions.

3.4

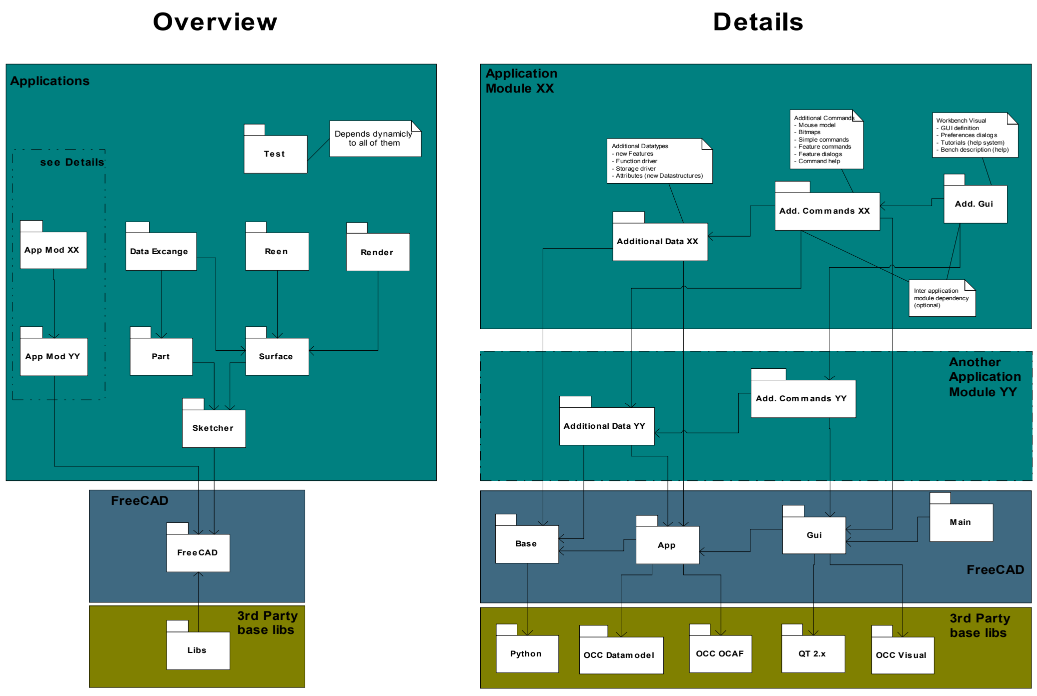
4 Design



In opposite to the Specification the Design gives you a discription of the system from a programmers point of view. Here we got the description of the Packages and the key concepts of the Program. For that purpose also the UML is used.

4.1 Overview

The Application FreeCAD consists basically off following Packages:



As you see there are five bigger blocks:

- Third party libraries
- Console application
- Interactive application
- Additional data types and functions
- Additional application logic

Each block has his special function and use normally the functions on the blocks below. In the following section the main functions of the blocks and packages are described.

4.1.1Third party libraries

This are libraries which are not changed in the FreeCAD project. They are basically used unchanged as a dynamic link library (*.so or *.dll). If there is a change necessary or a wrapper class is needed, then the code of the wrapper or the changed library code have to be moved to the FreeCAD base package.

The used libraries are:

Python

Python is the primary scripting language and is used in the whole application.

For example:

- Implement test scripts for testing on:
  - memory leaks
  - ensure presents of functionality after changes
  - post build checks
  - test coverage tests
- Macros and macro recording
- Implement application logic for standard packages
- Implementation of whole workbenches
- Dynamic loading of packages
- Implementing rules for design (Knowledge engineering)
- Doing some fancy Internet stuff like work groups and PDM
- And so on ...

Especially the dynamic package loading of python is used to load at run time additional functionality and workbenches needed for the actual tasks. For a closer look to python see: www.python.org Why Python you may ask. There are some reasons: So far I used different scripting languages in my professional life:

- Perl
- Tcl/Tk
- VB
- Java

Python is more OO then Perl and Tcl, the code is not a mess like in Perl and VB. Java isn’t a script language in the first place and hard (or impossible) to embed. Python is well documented and easy to embed and extend. It is also well tested and has a strong back hold in the open source community.

OCC (Open CasCade)

OCC is a full featured CAD Kernel. Its originally developed by Matra Datavision in France for the Strim (Styler) and Euclid Quantum applications and later on made Open Source. It’s a really huge library and makes a free CAD application possible in the first place, by providing some packages which would be hard or impossible to implement in a Open Source project:

- A complete STEP compliant geometry kernel
- A topological data model and all needed functions to work on (cut, fuse, extrude, and so on. . .)

- Standard Import- / Export processors like STEP, IGES, VRML
- 3D and 2D Viewer with selection support
- A Document and Project data structure with support for save and restore, external linking of documents, recalculation of design history (parametric modelling) and a facility to load new data types as a extension package dynamically

To learn more about Open CasCade take a look at http://www.opencascade.org.

QT (2.x +)

I don't think I need to tell a lot about QT. Its one of the most often used GUI toolkits in Open Source projects. For me the most important point to use QT is the QT Designer and the possibility to load whole dialog boxes as a (XML) resource and incorporate specialized widgets. In a CAX application the user interaction and dialog boxes a by far the biggest part of the code and a good dialog designer is very important to easily extend FreeCAD with new functionality. Further information and a very good online documentation you'll find on http://www.troll.no.

4.1.2Console application

This is a executable which incorporates two important packages:

- FreeCAD Base
- FreeCAD Application

The console application is mainly for automated testing purpose, but can also be used for server services. The packages used in this application are completely GUI independent. The executable runs without a X-Server and is able to load additional function (trough python packages) and additional data types (trough OCC plugins)

4.1.3Interactive application

One package more on top of the Console Application makes the Interactive Application.

- FreeCAD Gui

This package brings the 3D and 2D Views, interactive selection, a tree view on the document and the framework to load workbenches. The Interactive application includes no modelling function, only helper function and the basic views.

4.2Key Pattern

In this Part I'w discribe the key design pattern used in FreeCAD. They stands for a the most important features of FreeCAD, as descriped in the Specification. Here a short introduction:

- Workbenches
  Deals with the GUI outline. Handels layout of Toolbars, Menues, Dockwindows and the Command Bar
- Features
  Handels the whole parametric, asociativ modeling. Parameter, recalculations and so on.
- Parameters
  Loading and saving user- and system parameters. Changing them in Dialog Boxes and Preferences Pages.

And here a more detailed description of the design:

4.2.1Workbenches

A Workbench is a object which mainly defines the whole outlouck of a User Interface.

4.2.2Features

This pattern incorporate the whole functionality of a parametric, asociativ modeling. The UML Layout of the Design you find in the Design Draftings:

- FCApp Feature S
  Is a layout of the most important classes for the Feature pattern.
- FCApp Feature D
  Is a dynamic view on the instanciated object in a document tree. Here you see the runtime layout of the most imported classes

5 Extending FreeCAD

5.1 Introduction

One of the most importent key stone of FreeCAD are the Application modules. FreeCAD itself is only a kind of library to allow Application modules to introduce Commands and Features. All build in functions are also Application modules!

There are diferent ways to extend FreeCAD. Easy ways and more powerfull approches. Here a short introduction:

1. Macro recording
2. Python programming
3. C++ / Python programming

Every of the above methodes has its specific advantages and disatvantages.

Macro recording is by far the easiest way to create new functions, but limitid to existing features. Its only possible to assamble existing features and functions to new and more powerfull commands. Its not possible to create new Features and Datatypes.

Python gives you much more possibilities. Its possible to create new Features and Commands, using all functions exportet from FreeCAD. In addition to this its also possible to use all the python packages out there. And that are realy a lot. From ZIP lib to more sophisitcated ones like whole webserver and webservices.

But the most powerfull extension you can only build with C++. There you can use every feature of OpenCasCade, FreeCAD and additional third party libraries. Also C++ is the right choice if you whant to make very time consuming algorithems.

5.2Macro recording

5.3Python progrmming

5.4Extending with C++

5.4.1Application modules

5.4.2Features

5.4.3View Provider

View Provider are resposibel for the visualization of a (new) Feature. It also manage the graphical interaction (editing) for a special or group of Features.

5.4.4Branding

Branding is easy speaking a way to hide FreeCAD into a simpler Application. Offten its not whanted to show the user the complexity of the whole FreeCAD and all its Workbenches. Its a good Idea to show him e.g. only one special Workbench which is needed for the purpose of the Application.