Osdag detailed work report for CAD & IFC development

annual work report

submitted by

Siddhesh S. Chavan

Prof. Siddhartha Ghosh

Principal Investigator

Department of Civil Engineering

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

March 2018

Chapter 1

CAD development

1.1 Brief about CAD modelling

The geometric kernel used in Osdag is *OpenCascade* where PythonOCC is considered to be the wrapper for the kernel. Every geometric kernel includes a modeling scheme. Similarly, the modeling scheme used in OpenCascade is Boundary Representation (BRep) which consists geometric primitives as vertex, edge, face, body and prism whereas operators used are boolean as add, subtract and union. This fundamental is used for CAD development in Osdag.

1.2 Flow of functions/variables for CAD model

As shown in Fig. 1.1, the python files in Component directory are basically the backbone of CAD model in Osdag. Each python file in Component directory is a class file which returns the 3D CAD model of whatever dimensions you pass to each python file viz. in a case of *plate.py* which includes a class object as *Plate* and it takes inputs as length (L), width(W) and thickness (T) of plate to create its CAD model. Therefore, whatever required dimensions you pass on to these class files, subsequent CAD models would get generated. This is the basic idea behind creating CAD models in Osdag.
Further, we will look at how the data from one python file to another file passes to create 3D CAD model with one example. The CAD model in Osdag is displayed, if and only if the design is safe else the graphics window stays dark.

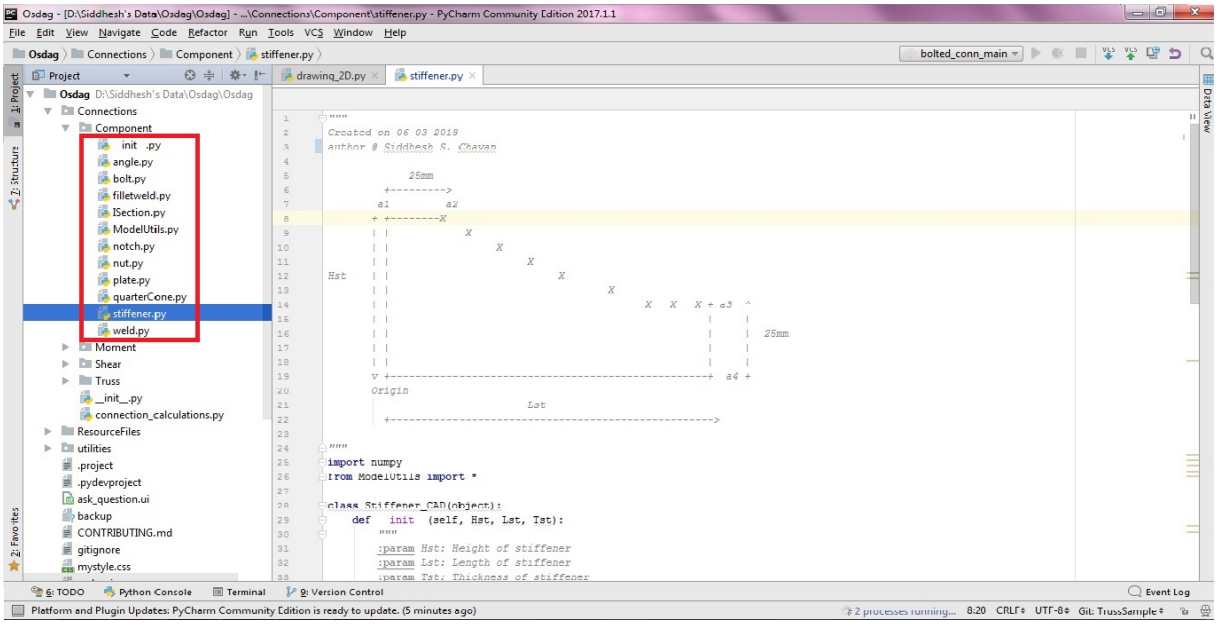


Figure 1.1: Expanded view of Component directory

1.3 Generalized flow of parameters

The generalized flow of variables/parameters is as discussed below;

1. When user clicks on Design button, all entered inputs are validated and component's dimensions are calculated in calculation file. Once, all calculations are carried out, compiler directs to a function, *def call_3DModel(self)* in main file of a module.

2. Initially, depending upon the safe or unsafe design it changes the graphics window color

3. In the same python file (Main_File) compiler moves to that function which contain the name of connectivity viz. ColFlangeBeamWeb, ColWebBeamWeb, Extended-BothWay etc.

4. This function basically initializes parameters for CAD model by taking UiObj (input dictionary) and OutputObj (output dictionary).

5. Next, the compiler moves to another python file with the same name as of connectivity name.

6. This file creates CAD model by calling the respective class files from component repository and returns the CAD model of individual components.

Above described steps would be best understood by applying breakpoint at *def call_3DModel(self)* in main file of a module and carrying out the step in debugging process.

1.4 CAD for moment connection of extended both way

The flow of parameters explaind at section 1.3, we'll illustrate it by taking an example of both way extended end plate module from moment connection. The below mentioned steps will display complete CAD model including welds at appropriate places in graphics window of Osdag.

1. Once user clicks on Design button from Input Dock, the debugger enters into the function named *def design_btnclicked()* in main file.
2. For creating CAD model, arrive at *def display_3DModel(self,component, bgcolor)*
3. Depending upon safe or unsafe design the background color in graphics window would change. Let's consider here as the design is safe.
4. Further, it calls the function named *def createExtendedBothWays()* for creating CAD model.
5. The first step here is to assign the beam properties from sqlite database depending upon the beam cross section entered by user.
6. Plate dimensions are pulled from output dictionary.
7. Bolt dimensions are decided on size of diameter selected by user.
8. For nut and bolt placement, it calls to *class NutBoltArray()* in *nutBoltPlacement.py* file
9. After acquiring all the data, it displays each model by *def get_model()* function.

Chapter 2

IFC development

2.1 Introduction

The Industry Foundation Classes (IFC) data model is intended to describe building and construction industry data. It is a platform neutral, open file format specification that is not controlled by a single vendor or group of vendors. It is an object-based file format with a data model developed by buildingSMART (formerly the International Alliance for Interoperability, IAI) to facilitate interoperability in the architecture, engineering and construction (AEC) industry, and is a commonly used collaboration format in Building information modeling (BIM) based projects. The IFC model specification is open and available. It is registered by ISO and is an official International Standard ISO 16739:2013.

2.2 IFC in Osdag

There are various IFC schema available and IFC file formats as IFC-SPF, IFC-XML and IFC-ZIP. However, here I have focused on IFC2X3 scheme and IFC-SPF file format. SPF stand here as **STEP Physical File**. FreeCAD can be used as an IFC viewer to view the CAD model of IFC whereas any editor as gedit, Notepad++ or sublime is capable of reading the IFC file.

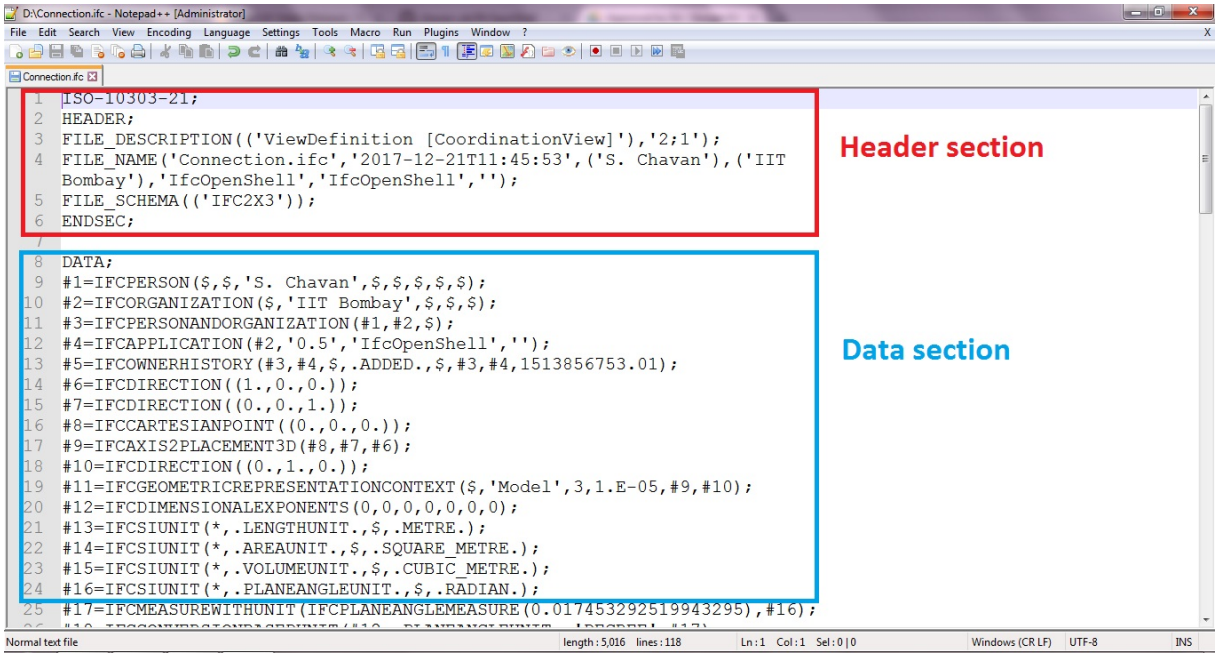


Figure 2.1: IFC file viewed in editor

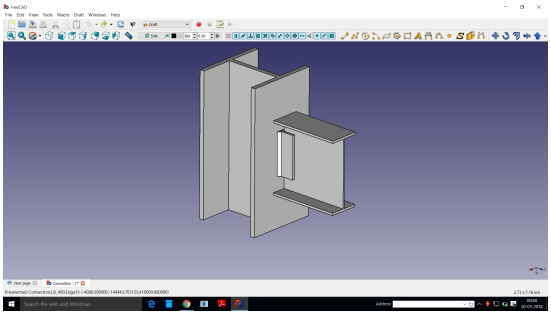
2.3 IfcOpenShell

The dependency required to generate IFC file is **IfcOpenShell.** To generate an IFC file of existing TopoDS CAD model, you need to have IFCOPENSHELL repository placed in site-packages. Don't try to install it through any python package manager such as conda or pip. Typically, any IFC file if you open it in any editor, it consist of 2 sections as header section and data section as shown in **Figure 2.1**.

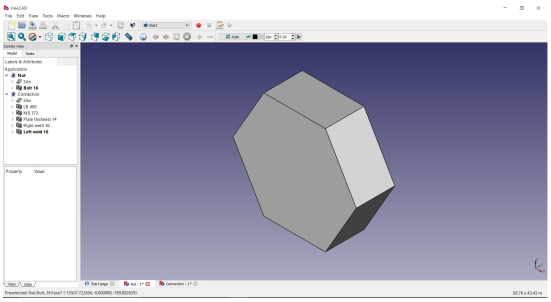
Header section consists of general description about IFC file such as IFC IFC_name, IFC_Schema utilized, which software is used to export the IFC file and so on. Whereas data section consists of vertices and IfcOpenShell geometric commands to create the model in IFC file format. The commands such as IFCDIRECTION, IFCCARTESIANPOINT, IFCAXIS2PLACEMENT3D are important to create an IFC CAD model.

2.4 IFC model for fin plate shear connection

Here is a sample code which creates a 3D IFC model of fin plate shear connection, **Figure 2.2(a)** which includes CAD models of beam, column, finplate and weld models. Since all these models are planar so it's simple to get the vertices of each model and to generate the CAD model.



(a) IFC model of fin plate connection



(b) IFC model of bolt head

Figure 2.2: IFC models in FreeCAD

2.5 Issues and future scope

The next challenge in completing the fin plate connection is to create the bolts and nuts placement. So, initially I started by creating the bolt head, **Figure 2.2(b)**. The same code of bolt head can be used to create the nut. However, to create circular cross sections in IFC is bit challengin. Therefore, to create the circular cross sections I raised an issue over github which initially started with how to create an IFC file and later discussion moved on over how to create the circular cross section. Therefore, as suggested by an IFC expert Thomas Krinjen, serialize and tesselate are the two available functions which are capable of creating cylindrical section. However, the area returned by any if these two functions is discrete in nature i.e. it's a tesselated or discrete area which will not work in our case. The required cylindrical area should be smooth and and fine in nature. So, this approach didn't solve our problem.

On deep research, I found that there is a function available in IfcOpenshell named **IfcCircleProfileDef** which takes in argument as radius and length of cylinder and returns the IFC model of cylinder. But the issue here is that this function is available for C++ dependency of IfcOpenshell and not for python dependency. So, the next task to find out a library or python package which is capable of calling C++ functions.

Seasnake is such a package availale for python which is capable of doing this task. I left working on IFC here. Now the next challenge would be to find a way by which we can implement seasnake library and would be able to call IfcCircleProfileDef function from C++ library of IfcOpenshell.

You can find the IfcCircleProfileDef function, here.You can also visit the forum on IFC,here.

Chapter 3

Hyperlinks to python files of CAD

3.1 Overview

This chapter contains the hyperlinks to python files and brief description about each file. Whenever we begin for CAD development of any new module in Osdag, the first step in CAD development is to prepare it's CAD model of connection using hard coded values, by hard coded I meant that we keep every dimension fixed and create a sample CAD model. Once, this is approved by sir and the calculation file of concerned module is ready then we start for proper python class files. Followings are the hard coded python files of CAD model;

1. Beam-beam extended both way
2. Beam-beam cover plate bolted
3. For Trusses,
    - Single angle
    - Double angle opposite side
    - Double angle same side
    - Channel truss
4. 2D drawing (front view) of single angle