Summer Fellowship Report

On

Developing Test module and integrating with TravisCI, GUI modification, Design Preference UI development and Crash Reporting

Submitted by

Nitin Singh

Under the guidance of

Prof. Siddhartha Ghosh
Civil Engineering Department
IIT Bombay

Under the Mentorship of

Deepthi Reddy
Project Research Associate

July 6, 2020

Acknowledgment

I would like to thank FOSSEE for providing me a platform to work on something I am very interested in. I am thankful to everyone who thought of having and involved in selection process based on screening tasks. I am grateful to be a part of team which promotes open source software.

I thank all the Osdag members, who are wonderful mentors and great team. I thank Sourabh Das (Project Research Associate), Ajmal Babu MS (Project Research Associate), Danish Ansari (Project Research Assistant), Yash Lokhande (Project Research Assistant), Darshan Viswakarma (Project Research Associate), Anand Swaroop (Project Research Associate), Anjali Jatav (Project Research Assistant) and whole team, who made us feel welcome and planned all the tasks meticulously during this period.

I am grateful that I got a chance to work under Prof. Siddhartha Ghosh, who took time to mentor us and monitored individual contributions as well.

Contents

Chapter 1

Introduction

1.1 Osdag Internship

Osdag internship is provided under the FOSSEE project. FOSSEE project promotes the use of FOSS (Free/Libre and Open Source Software) tools to improve quality of education in our country. FOSSEE encourages the use of FOSS tools through various activities to ensure availability of competent free software equivalent to commercial (paid) softwares.
The FOSSEE project is a part of the National Mission on Education through Infrastructure and Communication Technology(ICT), Ministry of Human Resources and Development, Government of India.
Osdag is one such open source software which comes under the FOSSEE project. Osdag internship is provided through FOSSEE project. Any UG/PG/PhD holder can apply for this internship. And the selection will be based on a screening task.

1.2 What is Osdag?

Osdag is Free/Libre and Open Source Software being developed for design of steel structures. Its source code is written in Python, 3D CAD images are developed using PythonOCC. Github is used to ensure smooth workflow between different modules and team members. It is in a path where people from around the world would be able to contribute to its development. FOSSEE's "Share alike" policy would improve the standard of the software when the source code is further modified based on the industrial and educational needs across the country.

Design and Detailing Checklist (DDCL) for different connections, members and structure designs is one of the important bi-products of this project. It would create a repository and design guide book for steel construction based on Indian Standard codes and best industry practices.

1.3 Who can use ?

Osdag is created both for educational purpose and industry professionals. As Osdag is currently funded by MHRD, Osdag team is developing software in such a way that it can be used by the students during their academics and to give them a better insight look in the subject.
Osdag can be used by anyone starting from novice to professionals. It's simple user interface makes it flexible and attractive than other software. Video tutorials are available to help get started. The video tutorials of Osdag can be accessed here.

Chapter 2

Development and Integration of Test module

I have created a Unit Testing module using which individual modules like FinPlateConnection, BasePlateConnection are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules. This module is then integrated with TravisCI with the help of a .yml script which automatically builds a pull request when it is first opened, and whenever commits are added to the pull request.

2.1 Development

Unit Testing module is created with the help of 'unittest'library present in python3. It also uses off-screen renderer to create the design for each input file without popping up the OCC Viewer each time a design would be created. All the modules to be tested are modified such that they do not contain any imports related to pyqt5. Checkout the code here.

```
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok
runTest (__main__.TestModules) ... ok

----------------------------------------------------------------------
Ran 69 tests in 446.572s

OK
```

Figure 2.1: Ouput of the Test module on local system.

2.2 Integration

Above Module Testing code interacts with travisCI using a .travis.yml file.This file specifies the programming language used, the desired building and testing environment (including dependencies which must be installed before the software can be built and tested), and various other parameters.Instead of using the native python environment offered by travis, a custom CONDA environment is created which is much faster than the native one and offer functionalities like using multiple python versions, installing texlive and caching.

Caching is used to decrease the execution time of testing, all the packages once installed will be cached for future use.Checkout the .travis.yml file.

To install Texlive on Travis server for creating design report pdf, a *.sh script is used along with a *.profile file. Checkout the .sh script here and .profile script here.

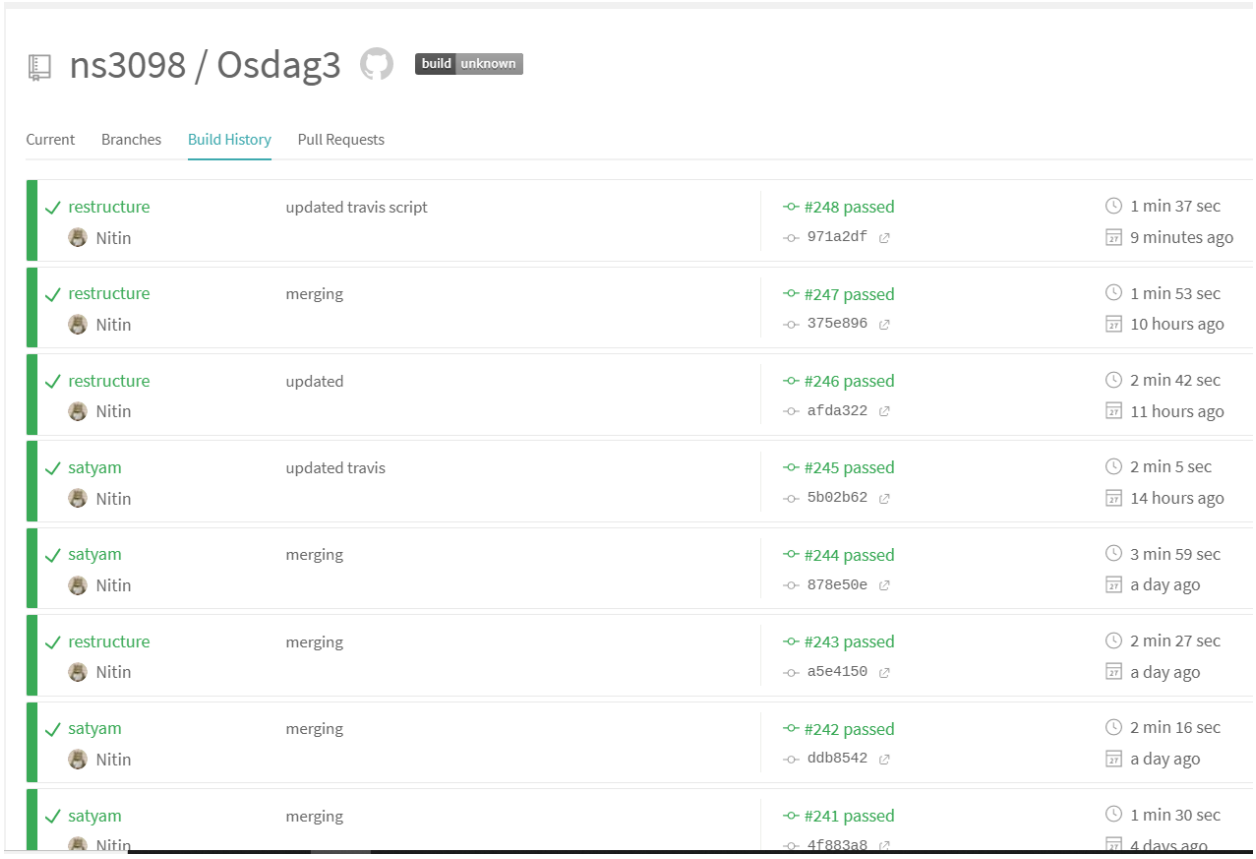


Figure 2.2: Output on travis Server.

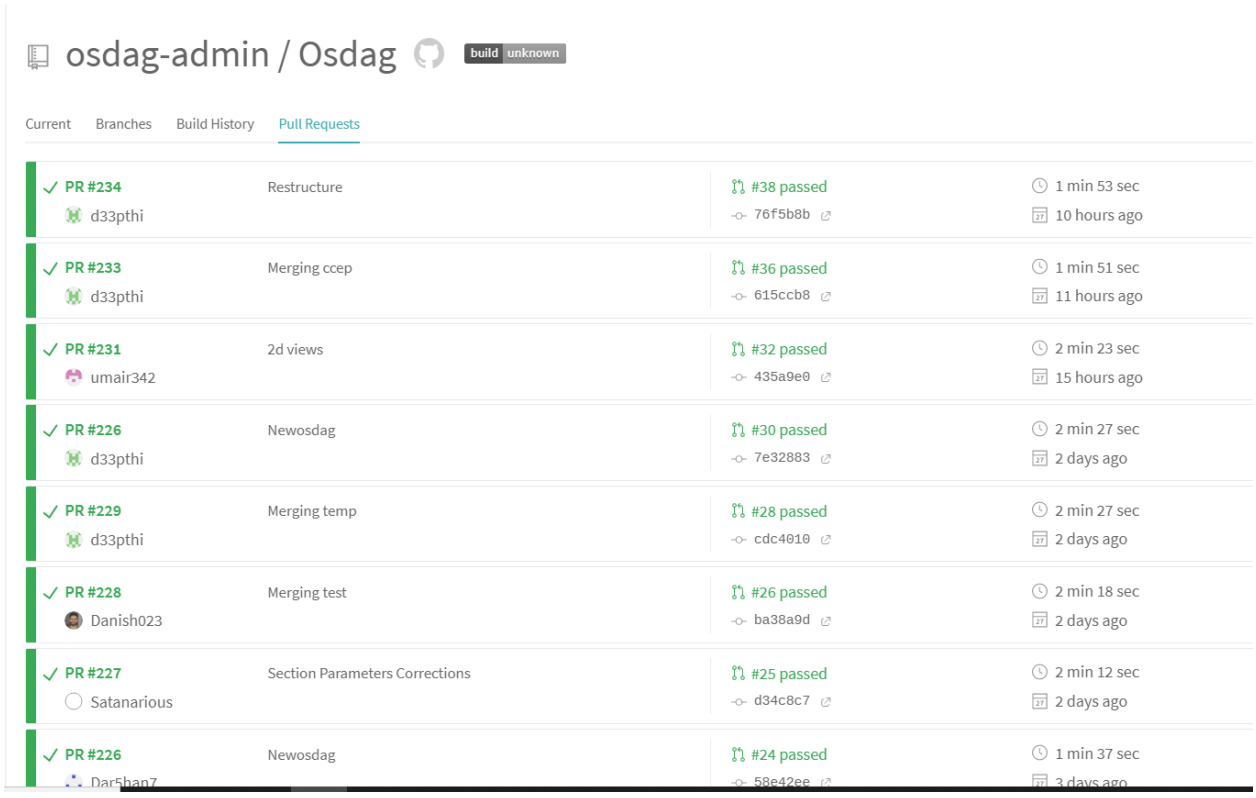


Figure 2.3: Pull requests build on travis Server.

Chapter 3

GUI modification

3.1 Input and Output Docks

Fixed the scaling of both the docks according to system resolution. Checkout the PR here and the original code here. (Check the new class added to resize the dock from line no. 106 to 152). All other changes are not at one place.

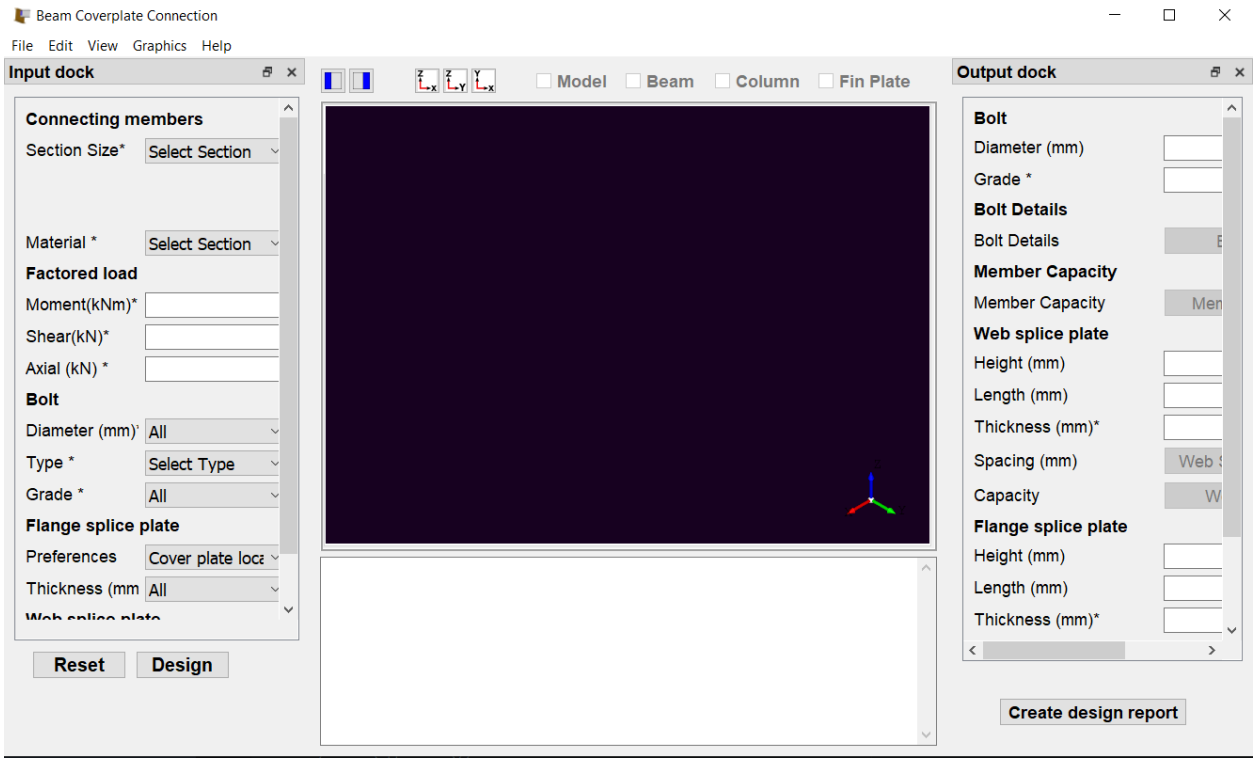


Figure 3.1: Before the modification.

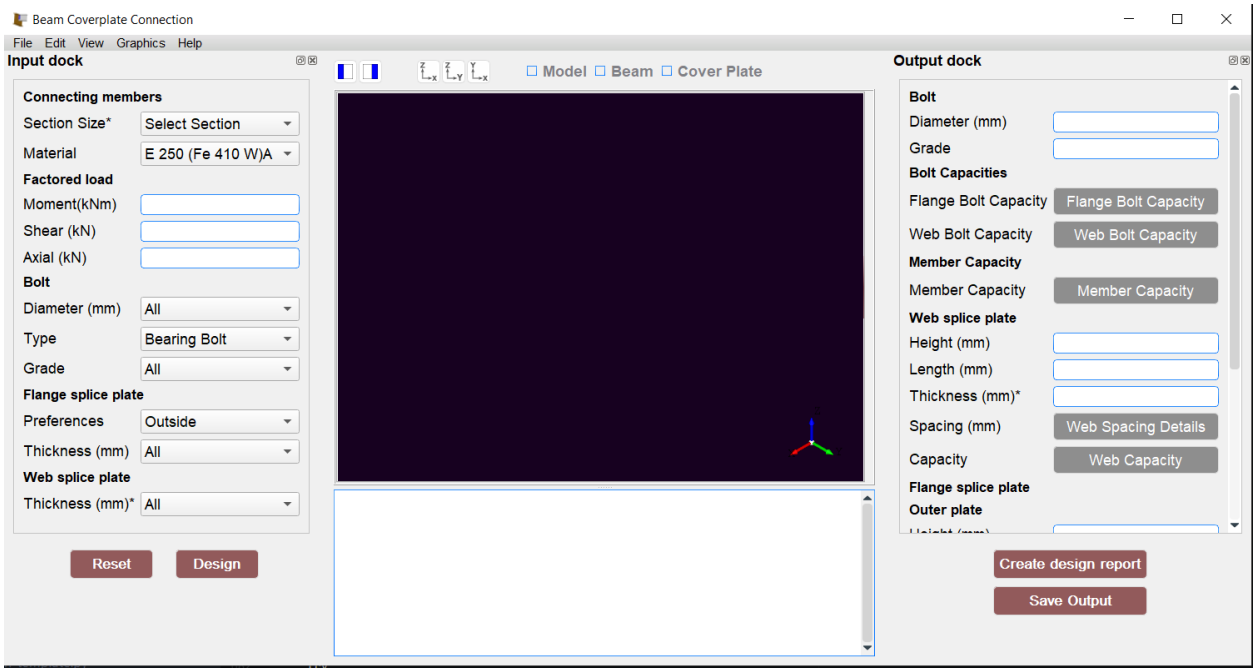


Figure 3.2: After the modification.

3.2 Save Output

Added a 'Save Output'button in the Output dock which saves all the data present in both Input and Ouput dock in a CSV file. Checkout the PR here and original code here. (from line no. 1375 to 1417)

	A	B	C	D
1	Module	Fin Plate	Bolt.Diameter	16
2	Connectivity	Column flange-Beam web	Bolt.Grade_Provided	8.8
3	Member.Supporting_Section.Designation	UC 356 x 406 x 393	Bolt.Shear	58.01
4	Member.Supported_Section.Designation	MB 500	Bolt.Bearing	72.38
5	Material	E 250 (Fe 410 W)A	Bolt.Capacity	58.01
6	Load.Shear	60	Bolt.Force (kN)	43.67
7	Load.Axial	40	Bolt.Line	1
8	Bolt.Diameter	['16']	Bolt.OneLine	2
9	Bolt.Type	Bearing Bolt	Bolt.Pitch	0
10	Bolt.Grade	['3.6', '4.6', '4.8', '5.6', '5.8', '6.8', '8.8', '9.8', '10.9', '12.9']	Bolt.EndDist	35
11	Connector.Plate.Thickness_List	['3', '4', '5', '6', '8', '10', '12', '14', '16', '18', '20', '22', '24', '26', '28', '30']	Bolt.Gauge	230
12	Member.Supporting_Section.Material	E 300 (Fe 440)	Bolt.EdgeDist	35
13	Member.Supported_Section.Material	E 250 (Fe 410 W)B	Plate.Thickness	12
14	Bolt.TensionType	Pretensioned	Plate.Height	300
15	Bolt.Bolt_Hole_Type	Standard	Plate.Length	80
16	Bolt.Slip_Factor	0.3	Plate.Shear	311.77
17	Weld.Fab	Shop Weld	Plate.Rupture	689.04
18	Weld.Material_Grade_OverWrite	410	Plate.BlockShear	340.54
19	Detailing.Edge_type	a - Sheared or hand flame cut	Plate.TensionYield	540
20	Detailing.Gap	10	Plate.TensionRupture	706.58
21	Detailing.Corrosive_Influences	No	Plate.BlockShearAxial	489.22
22	Design.Design_Method	Limit State Design	Plate.MomDemand	2.7
23	Connector.Material	E 165 (Fe 290)	Plate.MomCapacity	40.5
24	Bolt.Material_Grade_OverWrite	410	Weld.Size	10
25			Weld.Strength	937.62
26			Weld.Stress	204.98

Figure 3.3: CSV file.

3.3 Adding styles and themes

Created two themes for the application light and dark. Also added a toggle button to change the stylesheet. Fixed the resizing issue of Osdag header and images in popup window.
Checkout the dark theme here.
Checkout the light theme here.
PyQt5 does not offer a toggle button, so i created one of my own by inheriting the QAbstractButton class. Checkout the code here.

Above stylesheets are used in the application with the help of function *setStyleSheet*, this function is used with a QApplication variable i.e. app in Fig 3.4 .

*.qss file contains the styleSheets of each type of widgets. It can style the widget specifically and generally both. To style the widget specifically we need to give it an objectName then we can use it in our qss file to style it.

For example Tabs and button style of 'Design Preference' Dialog is different from OsdagMainPage and Module Window in light theme. We have used objectName of the Design Preference dialog to set the stylesheet. Note the use of # to interact with widget using its objectName. See Fig 3.5 and Fig 3.6.

But see Fig 3.7 and Fig 3.8 how we generally declared the stylesheet without using any particular objectName.

It is also advised to not declare the StyleSheet of any widget inside the main UI code, instead declare it inside the qss file using its objectName. If we specifically set the styleSheet inside the main UI code then the styles declared in qss file will have no effect on this widget.

```
if __name__ == '__main__':

    app = QApplication(sys.argv)
    path = os.path.join(os.path.dirname(__file__), 'themes', 'light.qss')
    file = QFile(path)
    file.open(QFile.ReadOnly | QFile.Text)
    stream = QTextStream(file)
    app.setStyleSheet(stream.readAll())
    app.setStyle('Fusion')
```

Figure 3.4: stylesheet applied over app variable which is of type QApplication

```
QDialog#DesignPreferences QTabWidget::tab-bar{
    alignment: left;
}
QDialog#DesignPreferences QTabBar::tab {
                        margin-right:2px;
                border-top-left-radius: 2px ;
                border-top-right-radius: 2px ;
                border-bottom-left-radius: 0px ;
                border-bottom-right-radius: 0px ;
                height: 28px;
                width: 122px;
                        background: lightgray;
                font-size: 14px;

                        color: black;
                        border: 0.5px inset #abc250;
               }

QDialog#DesignPreferences QTabBar::tab:!selected:hover
{
    /*border-top: 2px solid #ffaa00;
    padding-bottom: 3px;*/
    border-top-left-radius: 3px;
    border-top-right-radius: 3px;
    background-color: qradialgradient(cx: 0.5, cy: 0.5, radius: 2, fx: 0.5, fy: 1, stop: 0 #abc250, stop: 0.2 #abc250, stop: 0.4 rgba(211,211,211, 80));
}
```

Figure 3.5: Specifically setting the stylesheet of Design Preference Dialog (Check light.qss file)

```

QDialog#DesignPreferences QPushButton{
                color: black;
                background:lightgray;
                border: 1.2px inset #abc250;

                padding-left: 7px;
                padding-right: 7px;
                font-family: Arial;
                font-size:16px;

}

QDialog#DesignPreferences QPushButton:hover{
        background-color: qradialgradient(cx: 0.5, cy: 0.5, radius: 2, fx: 0.5, fy: 1, stop: 0 #abc250, stop: 0.2 #abc250, stop: 0.4 rgba(211,211,211, 80));
}
QDialog#DesignPreferences QTabBar::tab:selected
{
    /*border-top: 2px solid #ffaa00;
    padding-bottom: 3px;*/
    border-top-left-radius: 3px;
    border-top-right-radius: 3px;
    background-color: QLinearGradient(x1:0, y1:1, x2:0, y2:0, stop:1 lightgray, stop:0.4 lightgray, stop:0.2 lightgray, stop:0.02 #c28889);
        border:none;
}
```

Figure 3.6: Specifically setting the stylesheet of Design Preference Dialog (Check light.qss file)

```
QTabBar::tab:first:!selected
{
 margin-left: 0px; /* the last selected tab has nothing to overlap with on the right */


    border-top-left-radius: 3px;
}

QTabBar::tab:!selected
{

    border-bottom-style: solid;
    margin-top: 3px;
    color: black;
}

QTabBar::tab:selected
{
    border-top-left-radius: 3px;
    border-top-right-radius: 3px;
    margin-bottom: 0px;
    /*background-color: QLinearGradient(x1:0, y1:0, x2:0, y2:1, stop:1 #9eba30, stop:0.7 #FFFAFA, stop:0.87 #FFFAFA, stop:0.1 #FFFAFA);*/
        background-color: qradialgradient(cx: 0.5, cy: 0.5, radius: 2, fx: 0.5, fy: 1, stop: 0 rgba(146,90,91,190), stop: 0.2 #925a5b, stop: 0.4 rgba(255,30,30,32));
        color: white;
}

QTabBar::tab:!selected:hover
{
    /*border-top: 2px solid #ffaa00;
    padding-bottom: 3px;*/
    border-top-left-radius: 3px;
    border-top-right-radius: 3px;
    background-color: QLinearGradient(x1:0, y1:0, x2:0, y2:1, stop:1 whitesmoke, stop:0.4 whitesmoke, stop:0.2 whitesmoke, stop:0.1 #925a5b);
}
```

Figure 3.7: Generally setting the stylesheet for QTab Widgets and it'll be applied to all QTabs except those in Design Preference dialog because we are declaring separate stylesheet for Design Preference using its objectName (Check light.qss file)

```
QPushButton{
        border-style: solid;
        font-family: Arial;
        border-top-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 rgb(215, 215, 215), stop:1 rgb(222, 222, 222));
        border-right-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(217, 217, 217), stop:1 rgb(227, 227, 227));
        border-left-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(227, 227, 227), stop:1 rgb(217, 217, 217));
        border-bottom-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 rgb(215, 215, 215), stop:1 rgb(222, 222, 222));
        border-width: 0.7px;
        border-radius: 5px;
        padding-top:4.5px;
        padding-bottom:4.5px;
        color: white;
        background-color: #925a5b;
}
QPushButton::default{
        border-style: solid;
        border-top-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 rgb(215, 215, 215), stop:1 rgb(222, 222, 222));
        border-right-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(217, 217, 217), stop:1 rgb(227, 227, 227));
        border-left-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(227, 227, 227), stop:1 rgb(217, 217, 217));
        border-bottom-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 rgb(215, 215, 215), stop:1 rgb(222, 222, 222));

}
QPushButton:hover{
        border-style: solid;
        border-top-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 #E0D4AE, stop:1 rgb(222, 222, 222));
        border-right-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(197, 197, 197), stop:1 rgb(227, 227, 227));
        border-left-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(227, 227, 227), stop:1 rgb(197, 197, 197));
        border-bottom-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 rgb(195, 195, 195), stop:1 rgb(222, 222, 222));
        background-color: qradialgradient(cx: 0.5, cy: 0.5, radius: 2, fx: 0.5, fy: 1, stop: 0 rgba(146,90,91,190), stop: 0.2 #925a5b, stop: 0.4 rgba(255,30,30,32));
        color:white;
}
QPushButton:pressed{
        border-style: solid;
        border-top-color: qlineargradient(spread:pad, x1:0.5, y1:1, x2:0.5, y2:0, stop:0 rgb(215, 215, 215), stop:1 rgb(222, 222, 222));
        border-right-color: qlineargradient(spread:pad, x1:0, y1:0.5, x2:1, y2:0.5, stop:0 rgb(217, 217, 217), stop:1 rgb(227, 227, 227));
```

Figure 3.8: Generally setting the stylesheet for all QPushButton Widgets and it'll be applied to all QPushButtons except those in Design Preference dialog because we are declaring separate stylesheet for Design Preference using its objectName (Check light.qss file)

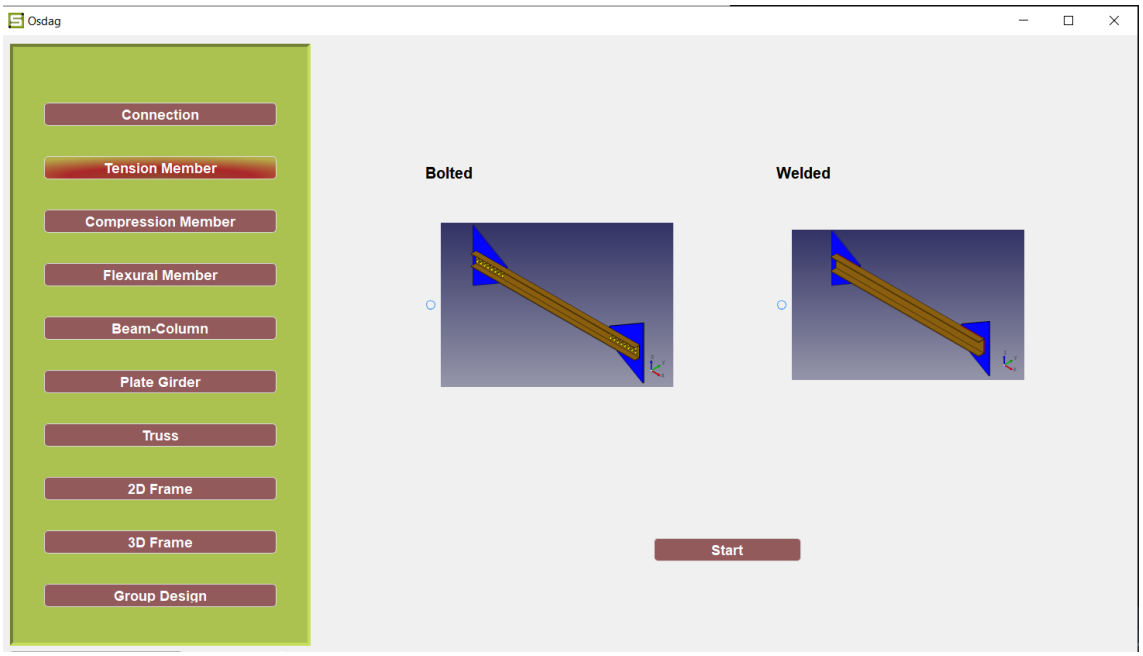


Figure 3.9: Easy to identify which tab is selected.

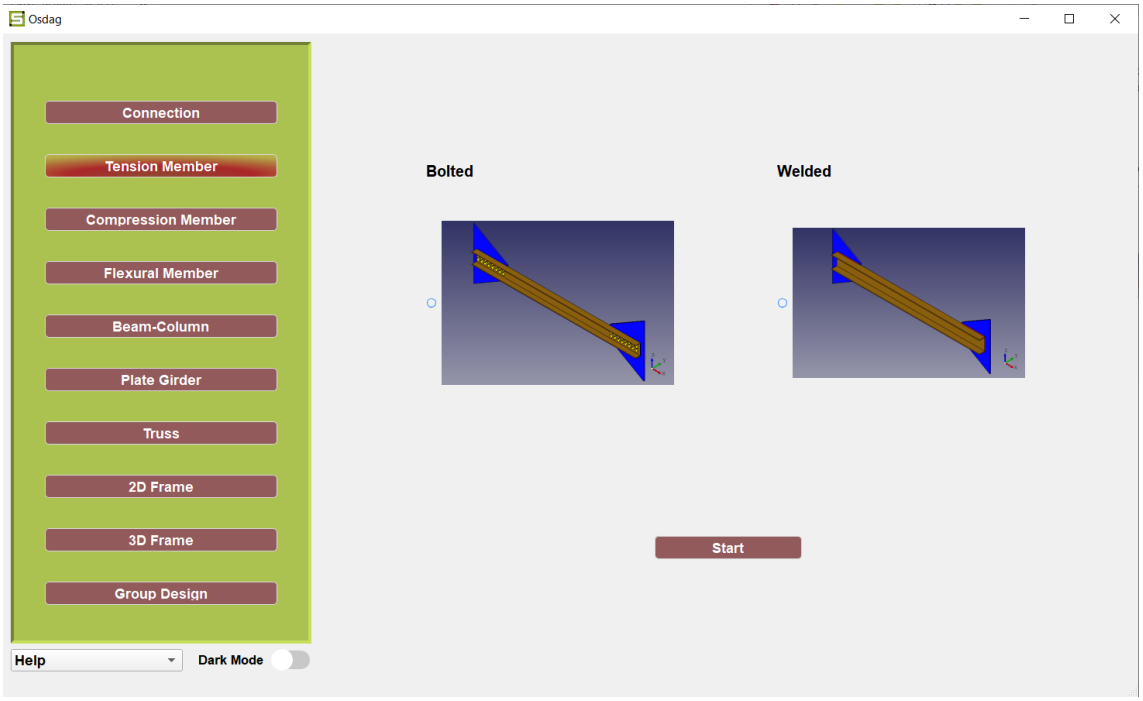


Figure 3.10: Toggle button at the bottom.

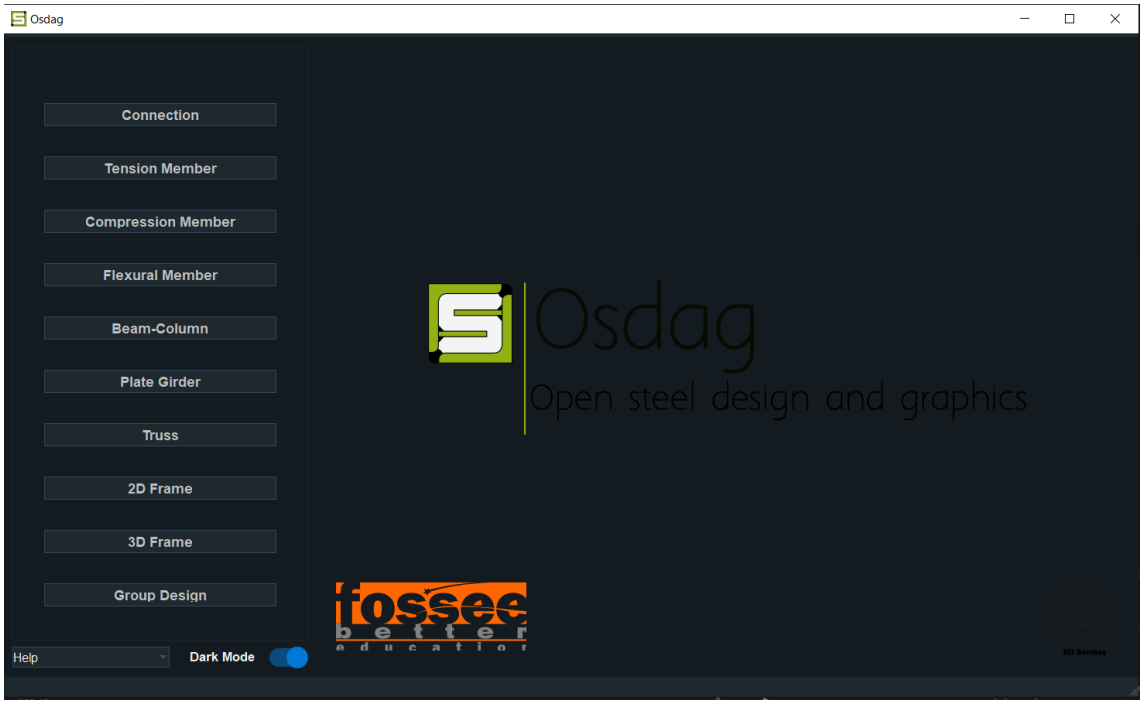


Figure 3.11: Dark Mode enabled.

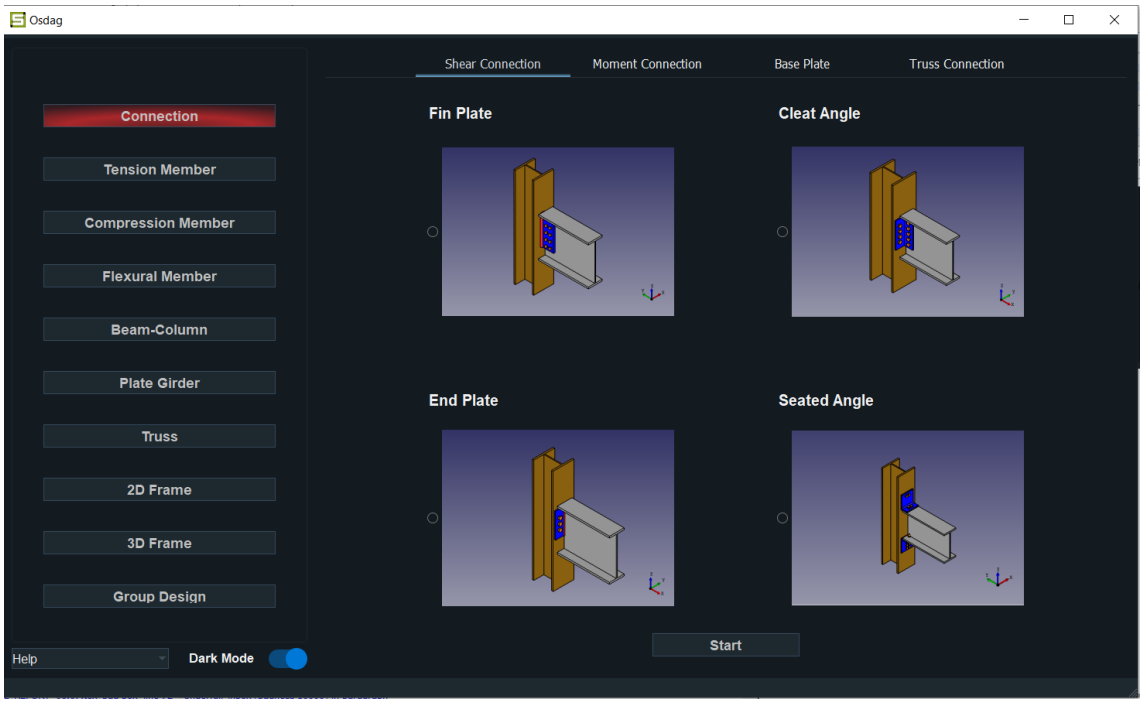


Figure 3.12: Dark Mode enabled.

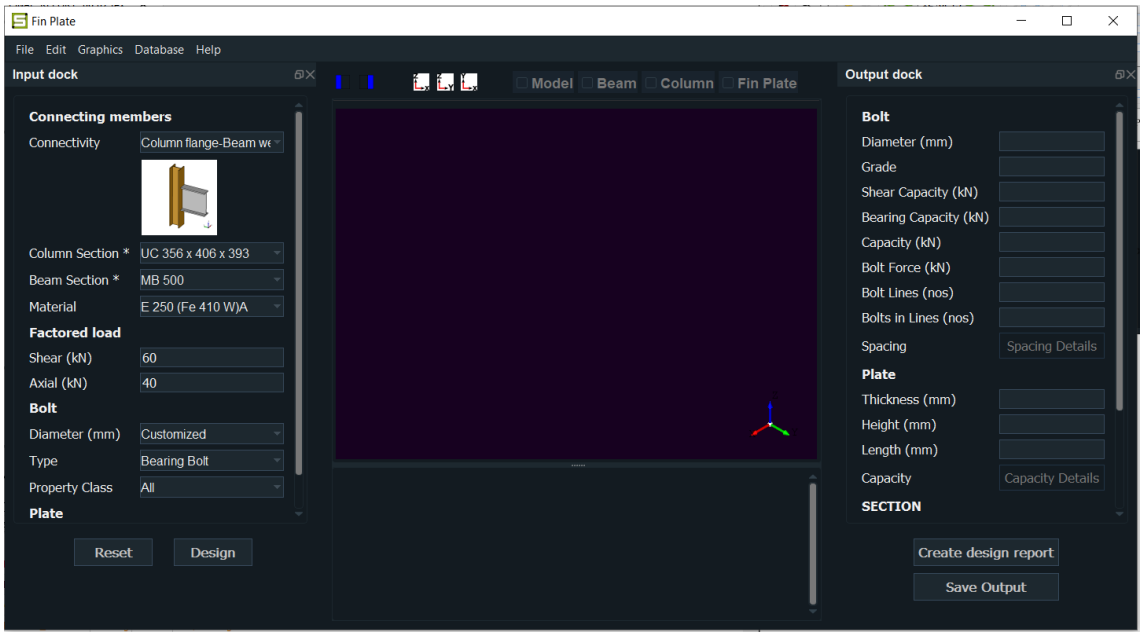


Figure 3.13: Dark Mode enabled.

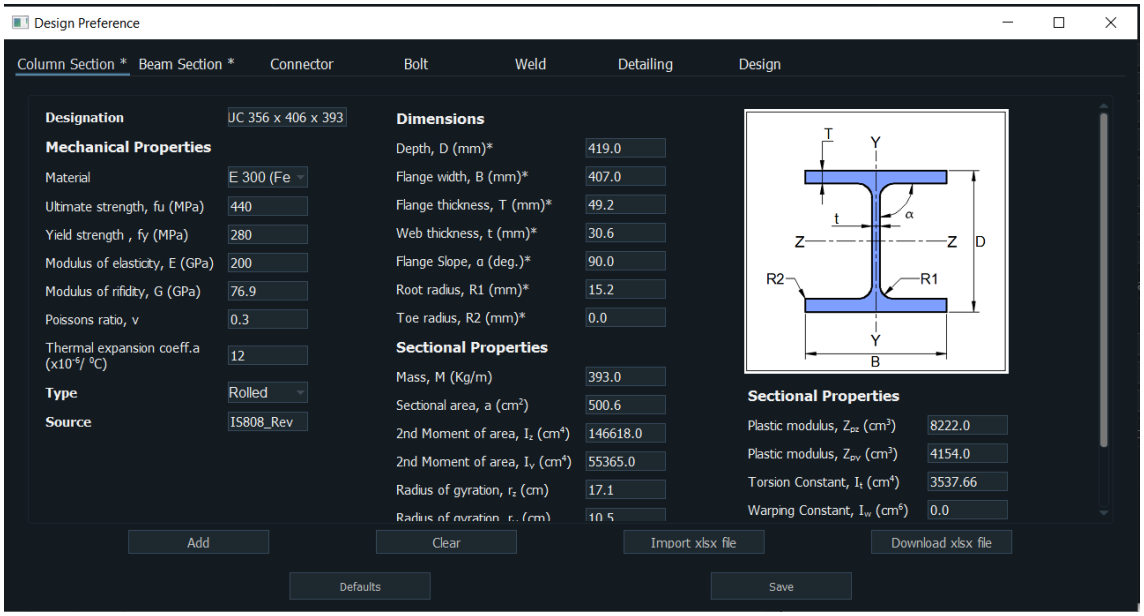


Figure 3.14: Dark Mode enabled.

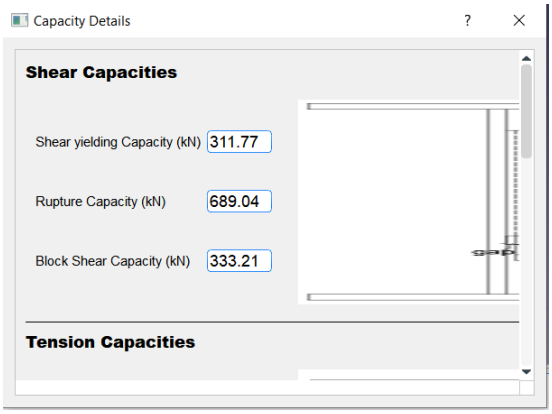


Figure 3.15: Before(Popup Window)

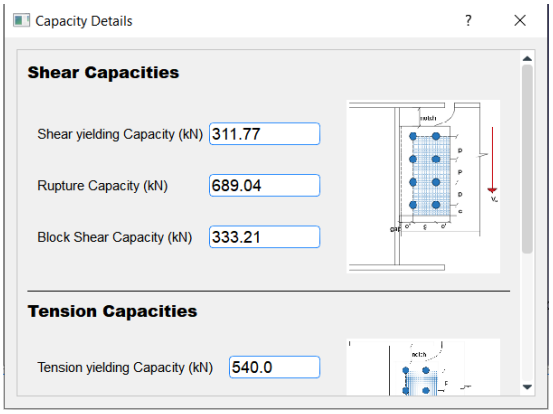


Figure 3.16: After(Popup Window)

Figure 3.17: Before(Osdag logo on lower resolutions)

Figure 3.18: After(Osdag logo on lower resolutions)

3.4 Working with widgets

How to adjust drop-down width of QCombobox according to length of longest text present?

This was not an issue till Qt4.8, If you had a QComboBox with a fixed size and added items which were longer than it, when the popup listview opened it would automatically resize to the longest string content.
But this doesn't work anymore with latest versions like Qt5.5+, this is a known issue.

Only way to deal with problem is to calculate the width of each item to be added in QCombobox and then set the width using minimumWidth, shown in the image below.

```
from PyQt5 import QtWidgets, QtGui

item_to_add = ['this', 'is', 'a', 'demo']  ''' list of items to be
                                            added in demo'''

combo = QtWidgets.QComboBox()

font = QtGui.QFont()
font.setPointSize(10)
font.setBold(False)
font.setWeight(50)
combo.setFont(font)

metrices = QtGui.QFontMetrics(font)  '''add font if it's applied on QCombobox'''

width = -1

for item in item_to_add:

    combo.addItem(item)

    width = max(width, metrices.boundingRect(item).width())  '''boundingRect gives most accurate result'''

combo.view().setMinimumWidth(width + 25) ''' Add some extra width to adjust the size of Scollbar
                                         when it shows up.'''
```

Figure 3.19: Demo to set the drop down width.

Image sizes on QLabel?

To control the size of image we need to use the function shown in the image below.

```
from PyQt5 import QtWidgets, QtGui

im = QtWidgets.QLabel()

pmap = QPixmap('path/to/image') # Enter the path to your image

#im.setScaledContents(1)
                        '''
                            Dont' use 'setScaledContents(1)' unless and until you don't want to
                            control the size of image.
                        '''

im.setPixmap(pmap.scaled(170,340,QtCore.Qt.KeepAspectRatio, QtCore.Qt.FastTransformation))
'''
Enter your own size instead of (170,340) and if you want to keep the aspect ration of
the image use 'KeepAspectRatio' just like it is used here, else use 'IgnoreAspectRatio'
'''
```

Figure 3.20: Demo to set the size of image.

How to avoid images from occupying bigger percentage in lower resolutions?

Simple answer is we can't unless and until we are ready to make the extra effort it requires, let me explain the reason.

The solution to this depends upon what we want to do in the low resolution. Do we want it to look the same, but without the larger images or do we want to display a different design? It may not be practical to use the same design for a lower resolution.

The first thing to do is to connect to the QApplication.desktop() resized event to detect that a screen resolution change has occurred. At this point we could either forward the event to all our widgets to resize and use different images, resize the images they have, or display different forms that you create for different screen sizes.

So if we want the images to be the same size proportionally to the screen resolution then the only options we have is to either reduce the image sizes on the fly, after detecting the lower screen resolution, or have a separate set of images to load, which would be much quicker

than the first option.

That being said it totally depends on developers whether they want to use different images for different resolutions or not.

How to resize window size according to screen resolution and move it to center of screen?

Always follow the order of using the function, first resize the window then move it to center, then in the last use self.show() to show the window.

```python
from PyQt5 import QtWidgets, QtGui

class Window(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()

        resolution = QtWidgets.QDesktopWidget().screenGeometry()  # Get the resolution
        width = resolution.width()    # find width
        height = resolution.height()  # find height

        self.resize(width*(0.85),height*(0.75))  # resize to 85% of width and 75% of height

        self.center()

        self.show()

    def center(self):
        frameGm = self.frameGeometry()
        screen = QtWidgets.QApplication.desktop().screenNumber(QtWidgets.QApplication.desktop().cursor().pos())
        centerPoint = QtWidgets.QApplication.desktop().screenGeometry(screen).center()
        frameGm.moveCenter(centerPoint)
        self.move(frameGm.topLeft())

```

Figure 3.21: Demo to resize and center any window.

*Should we use *.ui files to create the UI or Custom code everything from scratch?*

Every time we start a project with some graphical toolkit, one of the first conflicts happen with the decision of how to deal with the visual design and the widget layout: A graphical tool or custom coding?

This is a quite tricky/subjective question because most people will decide based on personal preference. These are some of the points to be considered -

Qt Designer

- Good
    1. Exploration. Discover what widgets are available, the names for those widgets, what properties you can set for each, etc.
    2. Enforces separation of UI logic from application logic.
- Bad
    1. If you need to add or remove widgets at run-time, you have to have that logic in code. I think it's a bad idea to put your UI logic in two places.
    2. Making changes to nested layouts. When a layout has no widgets in it, it collapses, and it can be really hard to drag and drop a widget in to the location you want.

Custom coding

- Good
    1. Fast if you are very familiar with Qt.
    2. Best choice if you need to add or remove widgets at run-time.
    3. Easier than Qt Designer if you have your own custom widgets.
    4. With discipline, we can still separate UI layout from behavior. Just put the part to create and layout widgets in one place, and the part to set signals and slots in another place.
- Bad
    1. Slow and confusing if you are new to Qt.
    2. Does not *enforce* separation of layout from behavior.

In summary, start with Qt Designer and let it take you as far as it can, then custom code everything from there.

Some Qt practices to follow

Below mentioned points are totally based on my personal experience and it comes from at least one real bug I encountered while working with Qt applications.

Specific Workarounds / Bugs

- Using QTimer.singleShot repeatedly can cause lockups.
- Avoid using QGraphicsView with QGLWidget.
- QGraphicsItems should never keep a reference to the QGraphicsView they live in. (weakrefs are ok).
- Raising exceptions inside QGraphicsItem.paint() can cause crashes. Always catch exceptions inside paint() and display a message rather than letting the exception proceed uncaught.
- Changing the bounds of QGraphicsItems without calling prepareGeometryChange() first can cause crash.

Practices for Avoiding Exit Crashes

- QObjects that reference their parent or any ancestor can cause an exit crash.
- The easiest way to avoid exit crashes is to call os._exit() before python starts collecting Qt objects. However, this can be dangerous because some part of the program may be relying on proper exit handling to function correctly (for example, terminating log files or properly closing device handles). At a minimum, one should manually invoke the atexit callbacks before calling os._exit().
- QGraphicsScene with no parent can cause an exit crash.
- QGraphicsItems that are not part of a QGraphicsScene can cause crash on exit.

General Programming Practices

- If you must use multi-threaded code, never-ever access the GUI from a non-GUI thread. Always instead send a message to the GUI thread by emitting a signal or some other thread-safe mechanism.
- Be careful with Model/View anything. TableView, TreeView, etc. They are difficult to program correctly, and any mistakes lead to untraceable crashing. Use Model Test to help ensure your model is internally consistent.
- Understand the way Qt object management interacts with Python object management and the cases where this can go wrong.See.
  - Qt objects with no parent are "owned" by Python; only Python may delete them.
  - Qt objects with a parent are "owned" by Qt and will be deleted by Qt if their parent is deleted.
  - Example of Core Dump.
- A QObject should generally not have a reference to its parent or any of its ancestors (weak references are ok). This will cause memory leaks at best and occasional crashes as well.
- Be aware of situations where Qt auto-deletes objects. If the python wrapper has not been informed that the C++ object was deleted, then accessing it will cause a crash. This can happen in many different ways due to the difficulty PyQt and PySide have in tracking Qt objects.
  - Compound widgets such as a QScrollArea and its scroll bars, QSpinBox and its QLineEdit, etc. (Pyside does not have this problem)
  - Deleting a QObject will automatically delete all of its children (however PyQt usually handles this correctly).

Chapter 4

Design Preference UI

Rewrote the whole UI code of Design preference dialog with improvements like adding SrollArea in each tab along with required layouts.Before the modification every widget was added according to x and y coordinates, which could have caused problems on systems with lower resolutions where widgets could go out of the screen. Checkout the code here.

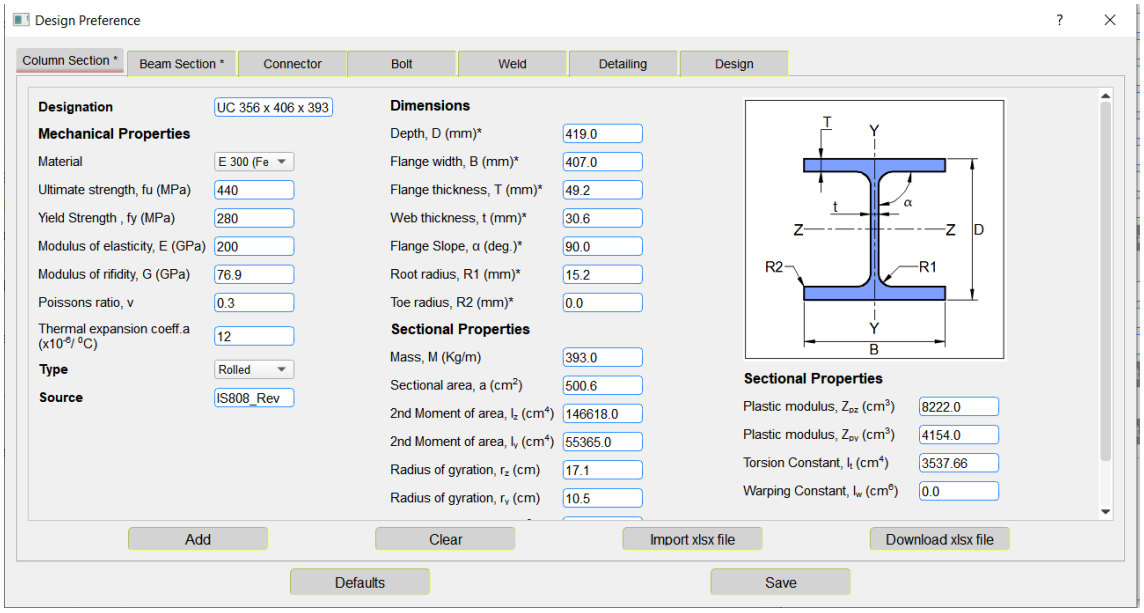


Figure 4.1: Design Preference dialog with layouts and ScrollArea.

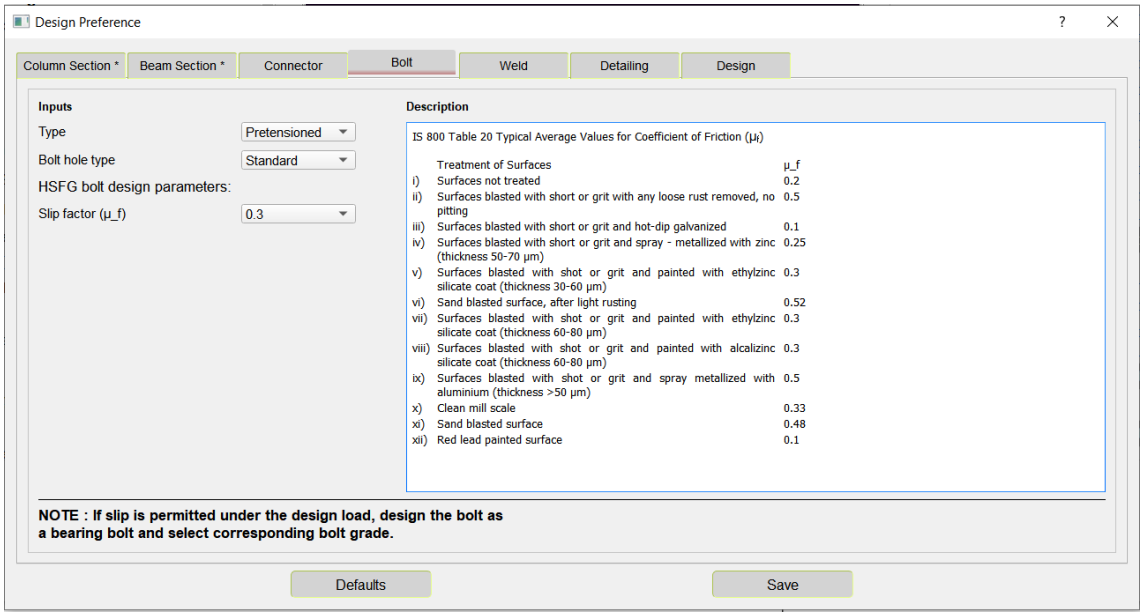


Figure 4.2: Design Preference dialog with layouts and ScrollArea.

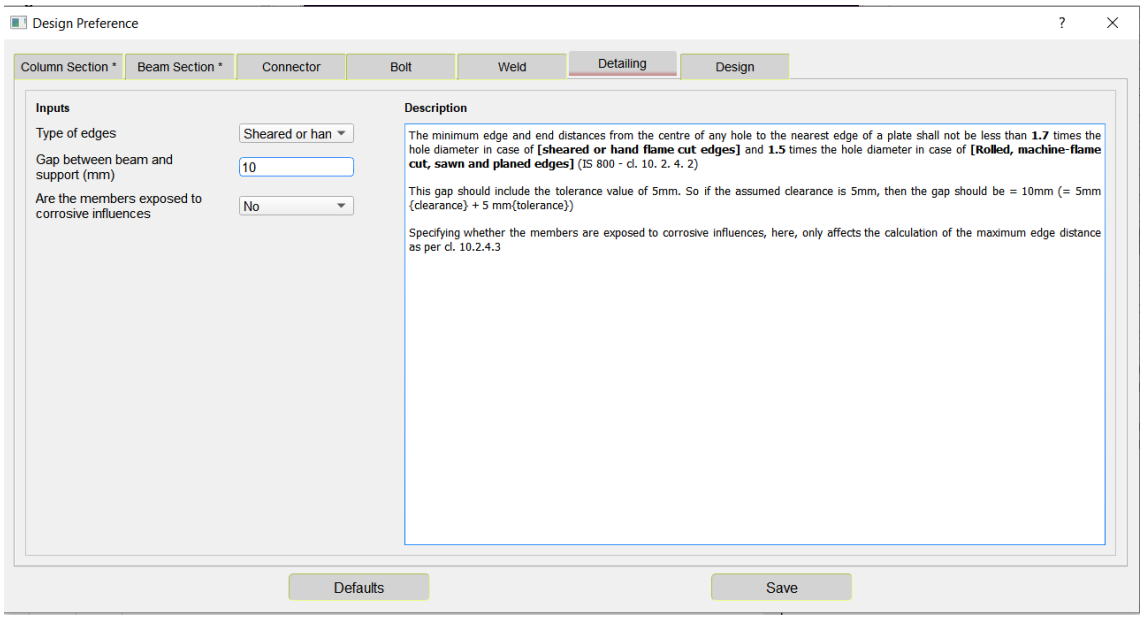


Figure 4.3: Design Preference dialog with layouts and ScrollArea.

Chapter 5

Crash Reporting

For most people, crashing means either an application has frozen or entirely disappeared. Sometimes, this is followed by a dialog box asking "Would you like to send a crash report?" Users hit "Send" and move on with their lives, and, most of the time, never think about that moment again.

But there is an entire world behind that interaction which is key to driving technology forward.

To break it down into layman's terms, a crash happens when a computer program fails to function properly and shuts down unexpectedly. Crashes happen for all kinds of reasons, but the main idea is that a program crashes when its code runs into a problem. In order to fix a crash, the code must be debugged, which is the process of finding and fixing the faulty code which caused the crash so the program can run smoothly again.

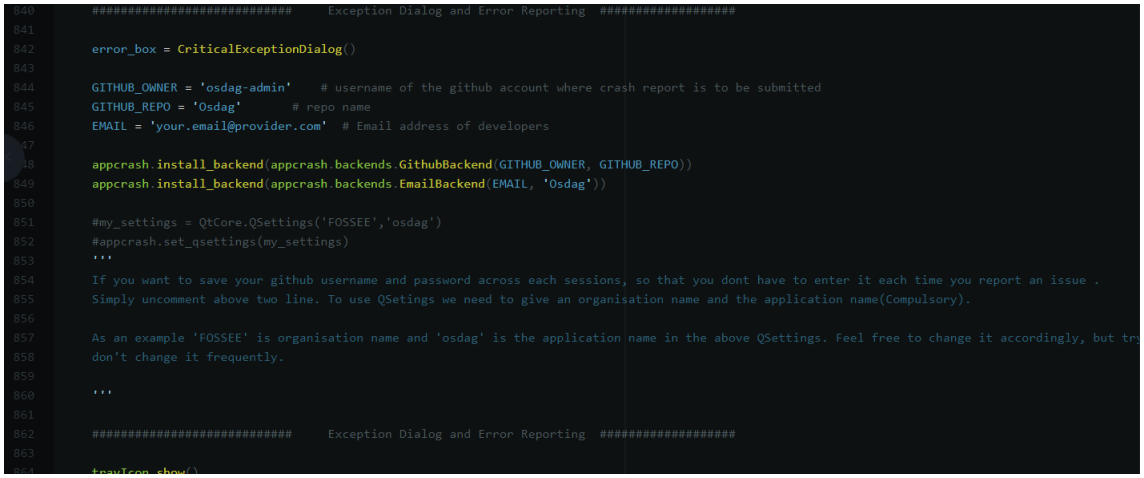
5.1 Exception Dialog

I have created a Dialog box which would appear immediately after application crashes due to some bugs or unhandled exceptions. It'll show all the information regarding the crash and also have two buttons in it "SAVE" and "REPORT ISSUE".

Clicking on "SAVE" button will write the whole crash report in a log file.

Checkout the Exception Dialog code here.

Checkout it's implementation code here.



```
    ############################     Exception Dialog and Error Reporting  ###################

    error_box = CriticalExceptionDialog()

    GITHUB_OWNER = 'osdag-admin'    # username of the github account where crash report is to be submitted
    GITHUB_REPO = 'Osdag'       # repo name
    EMAIL = 'your.email@provider.com'  # Email address of developers

    appcrash.install_backend(appcrash.backends.GithubBackend(GITHUB_OWNER, GITHUB_REPO))
    appcrash.install_backend(appcrash.backends.EmailBackend(EMAIL, 'Osdag'))

    #my_settings = QtCore.QSettings('FOSSEE','osdag')
    #appcrash.set_qsettings(my_settings)
    '''
    If you want to save your github username and password across each sessions, so that you dont have to enter it each time you report an issue .
    Simply uncomment above two line. To use QSetings we need to give an organisation name and the application name(Compulsory).

    As an example 'FOSSEE' is organisation name and 'osdag' is the application name in the above QSettings. Feel free to change it accordingly, but tr
    don't change it frequently.

    '''

    ############################     Exception Dialog and Error Reporting  ###################

    trayIcon.show()
```

Figure 5.1: Imported CriticalExceptionDialog and appcrash framework

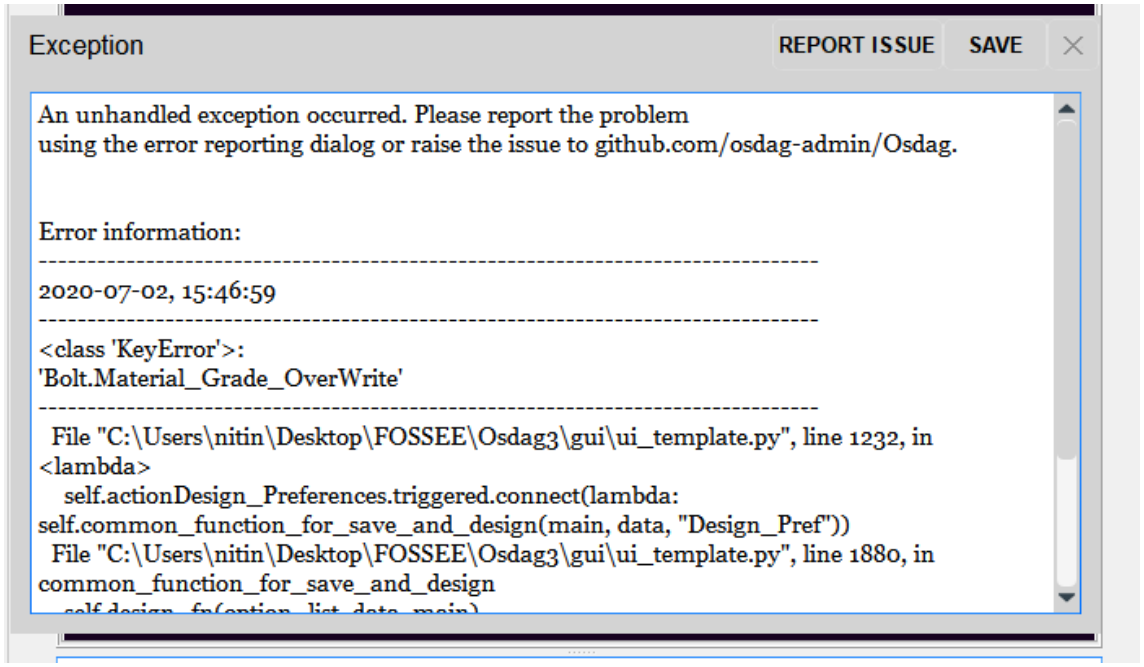


Figure 5.2: Exception Dialog

5.2 Report Issue

User can also report the issue by clicking on "REPORT ISSUE" button. I have created a framework for reporting application crash (unhandled exception) and/or let the user report an issue/feature request.

Some features of the framework:

1. Multiple builtin backends for reporting bugs:
   - GithubBackend: let you create issues on github.
   - Emailbackend: let you send an email with the crash report.
2. Highly configurable, you can create your own backend, set your own formatter,...
3. A thread safe exception hook mechanism with a way to setup your own function.

Steps involved in reporting the issue:

- Enter title and description of the issue.
- Review the report.
- Sign in to github using your username and password or Personal Access Token.
- Issue will be created on the application github repo, where developers can see it.

Checkout the whole framework code here.

Checkout it's implementation code here. (From line no. 840 to 862, 738 to 790)

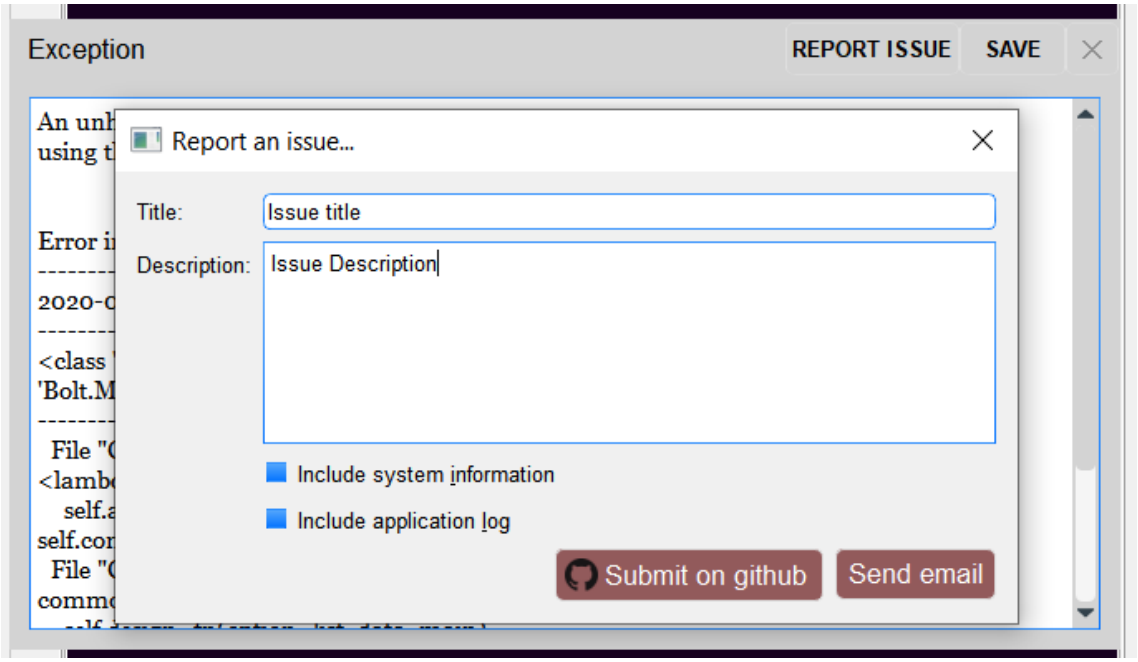


Figure 5.3: Report issue Dialog.

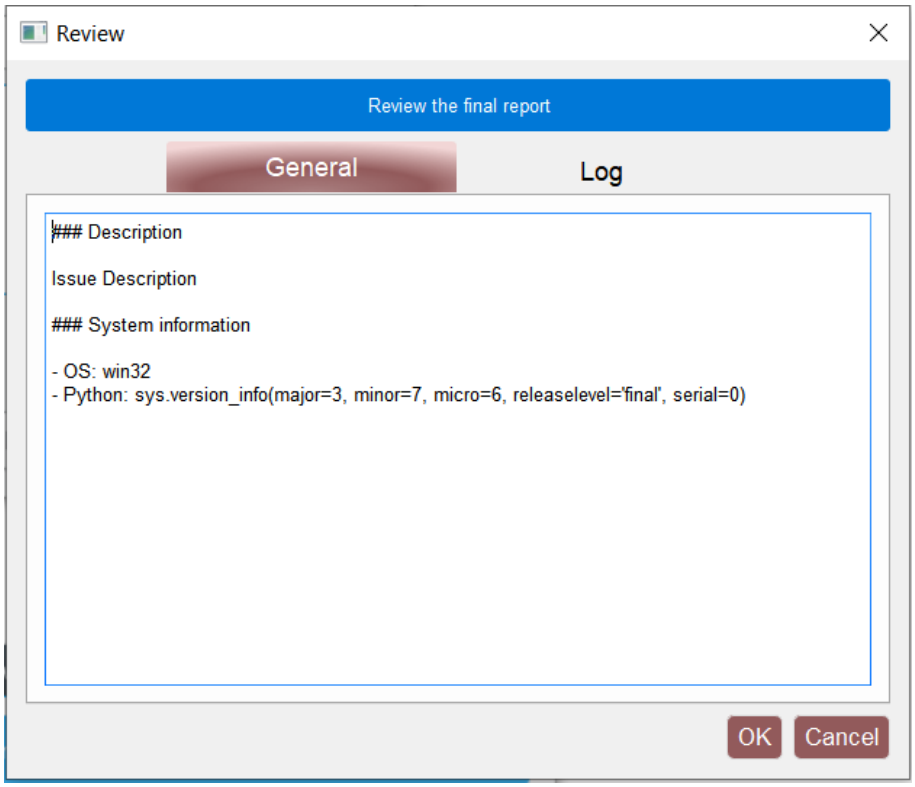


Figure 5.4: Review report before submitting.

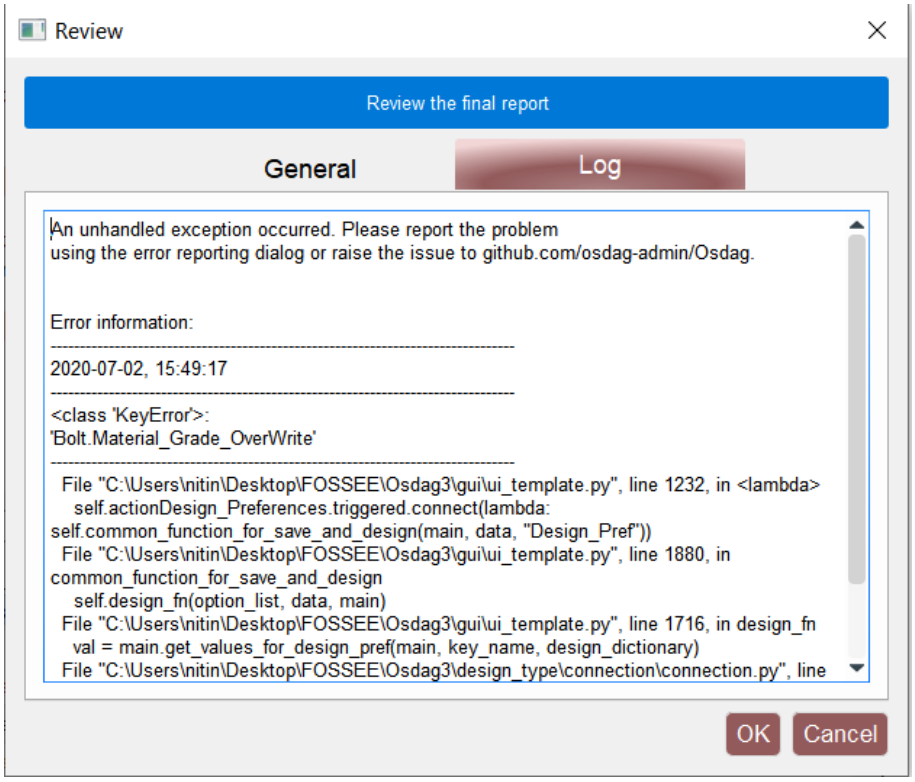


Figure 5.5: Review report before submitting.

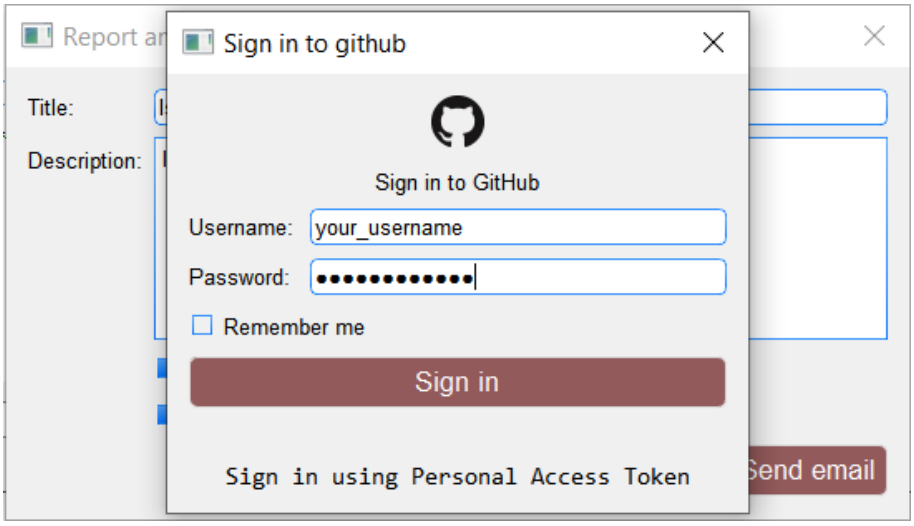


Figure 5.6: Basic authentication using Github username and password.

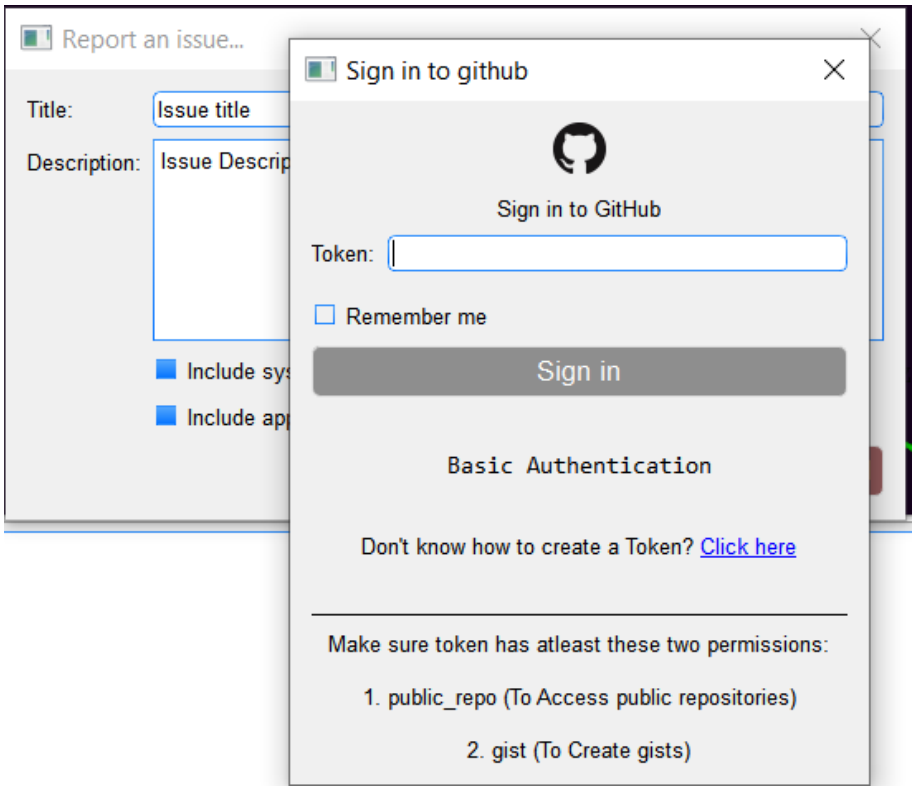


Figure 5.7: Sign in using personal access token.(In case user has enabled two factor authentication or basic authentication is not working.)

Chapter 6

Other modifications and Bug/Error Fixes

6.1 Wrapped C/C++ object of type QTextEdit has been deleted

This error was thrown because of creating multiple handler for the same logger without removing the previous ones while closing the module. Whenever any module opens a logging handle is created for this particular module which points to the QTextEdit box to show logs to the user. So when we close this module without removing this handler and open a new module, this particular handler is still there pointing to the address of QTextEdit box of the previously opened module which is actually closed(deleted) and hence this error shows up.
Solution was to remove all the created handlers while closing the module, so that a new handler can be created whenever we open another module pointing to new address of QtextEdit box.
Checkout the code here.

```
154     def closeEvent(self, event):
155         '''
156         Closing module window.
157         '''
158         reply = QMessageBox.question(self, 'Message',
159                                      "Are you sure you want to quit?", QMessageBox.Yes, QMessage
160 
161         if reply == QMessageBox.Yes:
162 
163             logger = logging.getLogger('osdag')  #  Remove all the previous handlers
164             for handler in logger.handlers[:]:
165                 logger.removeHandler(handler)
166 
167             self.closed.emit()
168             self.ui.designPrefDialog.close()
169             event.accept()
170         else:
171             event.ignore()
172 
```

Figure 6.1: Handlers removed here

6.2 Modifying Source Code

I have modified the souce code of python-occ to offer off-screen rendering for creating designs. It is used in Module testing to create design reports. Some useful links which helped me a lot for this modification: Link 1 Link 2 Link 3 Link 4 Link 5

Checkout the code here.