



# Summer Fellowship Report

On

Dynamic User Interface and New Features of Osdag

Submitted by

**Ansari Mohammed Umair**

Under the guidance of

**Prof. Siddhartha Ghosh**  
Civil Engineering Department  
IIT Bombay

and

**Mr. Sunil Shetye**  
Senior Project Manager  
FOSSEE

Under the Mentorship of

**Deepthi Reddy**  
Project Research Associate  
**Danish Ansari**  
Assistant Project Manager

July 4, 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 Deepthi Reddy (Project Research Associate), Sourabh Das (Project Research Associate), Danish Ansari (Assistant Project Manager), Yash Lokhande (Project Research Assistant), B Anand Swaroop Goud (Project Research Associate), Darshan Vishwakarma (Project Research Associate), Kumari 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 and Mr. Sunil Shetye, who took time to mentor us and monitored individual contributions as well.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Osdag Internship . . . . .	4
1.2	What is Osdag? . . . . .	4
1.3	Who can use? . . . . .	5
<b>2</b>	<b>Dynamic User Interface and New Features of Osdag</b>	<b>6</b>
2.1	Changes in Module_window UI . . . . .	6
2.1.1	Checkbox for 3d components . . . . .	7
2.1.2	Save 3d models and Cad images . . . . .	7
2.1.3	Custom Material Popup . . . . .	7
2.1.4	Download and Reset Database . . . . .	7
2.1.5	Load Previous Inputs . . . . .	7
2.1.6	Help options in module_window . . . . .	8
2.1.7	Zoom-in, Zoom-out, Pan and Rotate Options . . . . .	9
2.1.8	Output Button Popup . . . . .	9
2.1.9	Browse, Save and Load profile . . . . .	10
2.2	Design Preferences . . . . .	11
2.2.1	Download and Import buttons . . . . .	11
2.2.2	Import Validation . . . . .	11
2.2.3	New features . . . . .	11
2.3	Documentation . . . . .	12
	<b>Appendices</b>	<b>13</b>
<b>A</b>	<b>Code for Checkbox for 3d components</b>	<b>14</b>
<b>B</b>	<b>Code for Save 3d models and cad images</b>	<b>15</b>
<b>C</b>	<b>Code for Custom Material Popup</b>	<b>17</b>

<b>D</b>	<b>Code for Download and Reset Database</b>	<b>19</b>
<b>E</b>	<b>Code for Loading previous Inputs</b>	<b>21</b>
<b>F</b>	<b>Code for Help Options</b>	<b>23</b>
<b>G</b>	<b>Code for Zoom-in, Zoom-out, Pan and Rotate options</b>	<b>25</b>
<b>H</b>	<b>Code for Output Button Popup</b>	<b>27</b>
<b>I</b>	<b>Code for Browse, Save and Load profile</b>	<b>31</b>
<b>J</b>	<b>Code for Download and Import buttons</b>	<b>33</b>
<b>K</b>	<b>Code for Import validation</b>	<b>37</b>
<b>L</b>	<b>Code for New features</b>	<b>39</b>

# 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. And the selection was based on an interview through the job fair followed by a python test.

### 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.

### 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

# Dynamic User Interface and New Features of Osdag

Software User Interface is made Dynamic so that new modules can be added to the software without changing the UI code and hence taking remote contributions for the software becomes easy. The new features added to the software increase the usability of software.



Figure 2.1: Osdag Mainpage

### 2.1 Changes in Module\_window UI

Changes done in the ui of module\_window are for making the ui dynamic and adding some new features.

### 2.1.1 Checkbox for 3d components

I have created a function `get_3d_components` in module files to get the number of checkboxes to be shown in ui based on the module. These checkboxes are used to show different components of the design in 3d viewer. Concerned code is attached vide [Appendix-A](#).

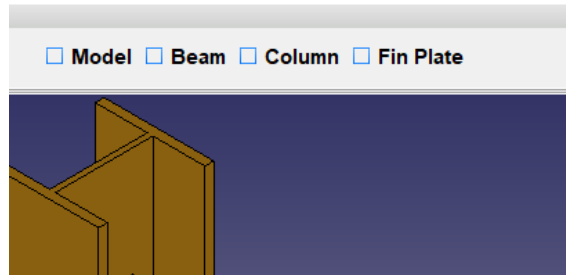


Figure 2.2: Checkbox

### 2.1.2 Save 3d models and Cad images

I have connected function for these options in menubar to save the created design as 3d model in any of (.brep, .stl, .step etc.) formats or as an image in any of (.jpg, .png etc.) formats. Concerned code is attached vide [Appendix-B](#).

### 2.1.3 Custom Material Popup

This popup allows user to add customized material grade in database and use that material grade for design. Concerned code is attached vide [Appendix-C](#).

### 2.1.4 Download and Reset Database

I have added a tab Database in menubar which gives users option of Downloading the Osdag database as an excel sheet or Resetting the database to default. Concerned code is attached vide [Appendix-D](#).

### 2.1.5 Load Previous Inputs

This feature saves the input dock values in an osi file with name same as module name on click of design button. While loading the mod-



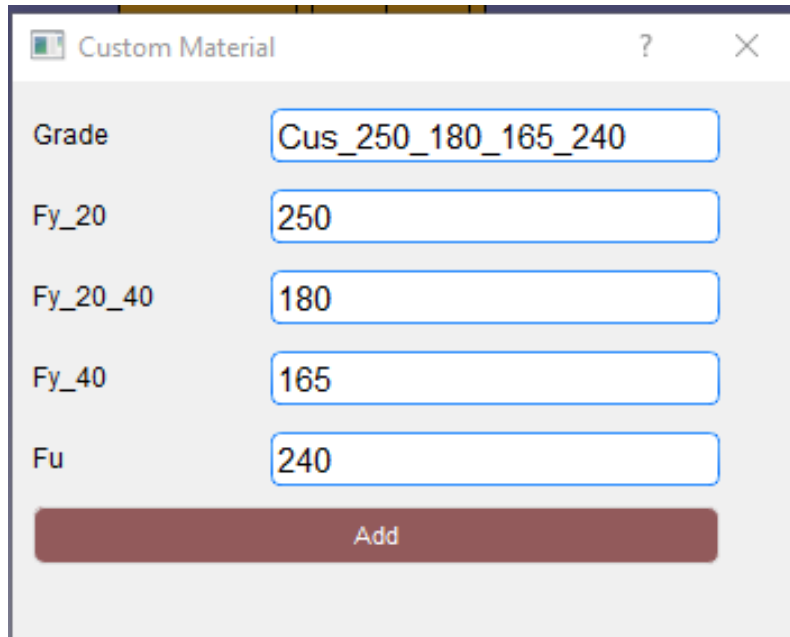


Figure 2.3: Custom Material Popup

ule\_window ui, if such osi file is present, then its values are loaded to the input dock. Concerned code is attached vide [Appendix-E](#).

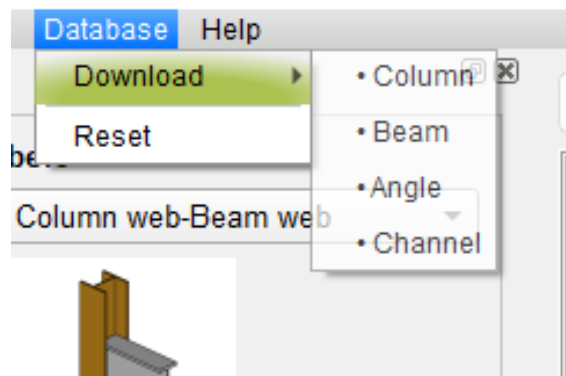


Figure 2.4: Download and Reset Database

### 2.1.6 Help options in module\_window

I have connected the Help menu options to their respective functions. Concerned code is attached vide [Appendix-F](#).

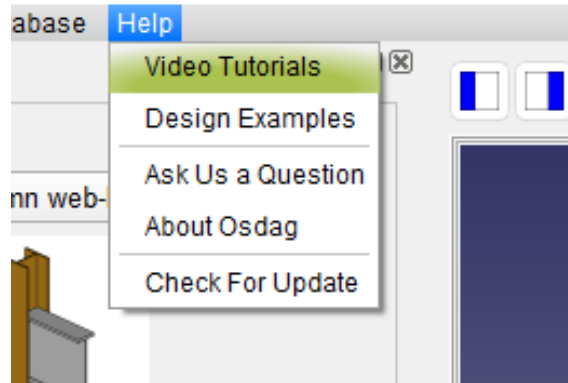


Figure 2.5: Help

### 2.1.7 Zoom-in, Zoom-out, Pan and Rotate Options

These options are used to change the size and positions of the 3d model created. I have also provided shortcuts for these options. Concerned code is attached vide [Appendix-G](#).

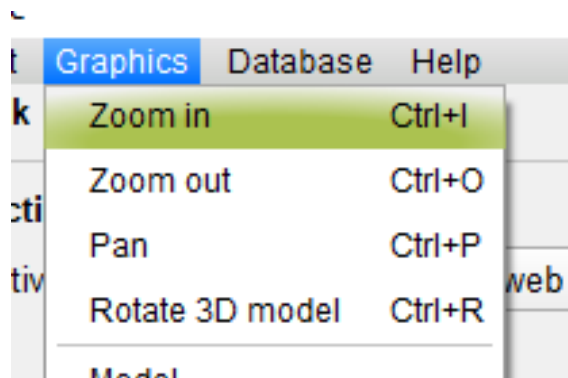


Figure 2.6: Zoom-in Zoom-out Pan and Rotate

### 2.1.8 Output Button Popup

I have created a function `output_button_dialog` which is used to show a dialog on click of `output_dock` button. Concerned code is attached vide [Appendix-H](#).

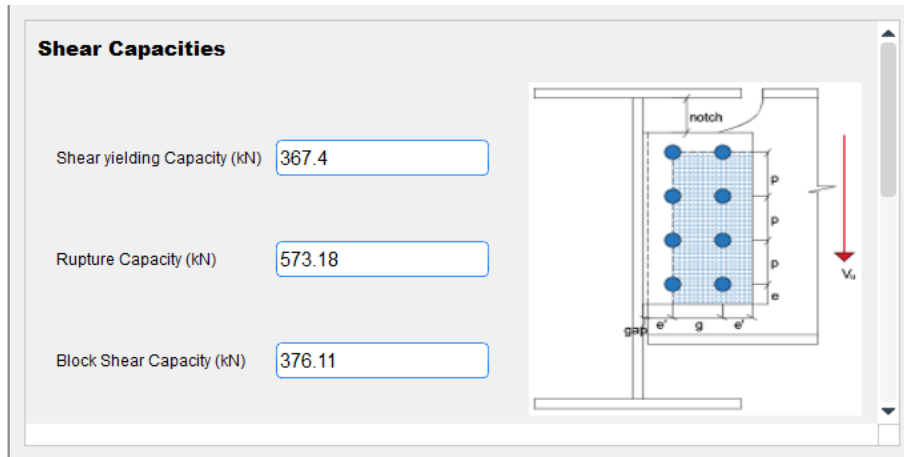


Figure 2.7: Output Button Popup

### 2.1.9 Browse, Save and Load profile

I have connected the functions for browse, save and load profile options in Design report popup. Concerned code is attached vide [Appendix-I](#).

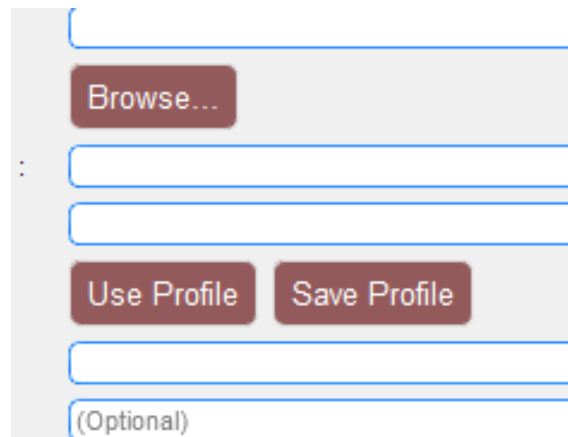


Figure 2.8: Browse, Save and Load profile

## 2.2 Design Preferences

I have connected design\_preference buttons to their functions and added some new features.

### 2.2.1 Download and Import buttons

Download option in design\_preferences can be used to download database header in an excel sheet. Import option is used to update the database using an excel sheet having same header as database. Concerned code is attached vide [Appendix-J](#).

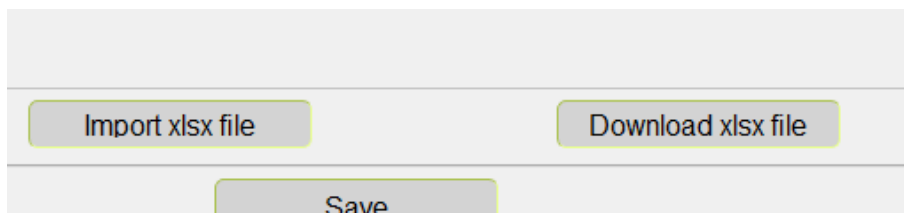


Figure 2.9: Download and Import

### 2.2.2 Import Validation

This feature validates the values of excel sheet to be imported in the database. Designations with invalid values are rejected and shown in a dialog box after valid designations are imported successfully. Concerned code is attached vide [Appendix-K](#).

### 2.2.3 New features

New features in design\_preferences include changing “Source” to “Custom” if designation is changed, adding validators to section dimensions and properties, making material properties non-editable, reducing extra keys and save design\_preferences changes only when design\_preferences is closed with “Save” button. Concerned code is attached vide [Appendix-L](#).

## 2.3 Documentation

I have created a Developer's Manual to help developers understand how the GUI code works. Developers manual includes documentation of how to create UI and generate log messages for new modules which will be useful for taking remote contributions through git.

# Appendices

## Appendix A

# Code for Checkbox for 3d components

```
1232
1233     #####
1234     # Function for individual component calls in 3D view
1235     #####
1236     def get_3d_components(self):
1237         components = []
1238
1239         t1 = ('Model', self.call_3DModel)
1240         components.append(t1)
1241
1242         t2 = ('Beam', self.call_3DBeam)
1243         components.append(t2)
1244
1245         t3 = ('Column', self.call_3DColumn)
1246         components.append(t3)
1247
1248         t4 = ('Fin Plate', self.call_3DPlate)
1249         components.append(t4)
1250
1251         return components
1252
1253     def call_3DPlate(self, ui, bgcolor):
1254         from PyQt5.QtWidgets import QCheckBox
1255         from PyQt5.QtCore import Qt
1256         for chkbox in ui.frame.children():
1257             if chkbox.objectName() == 'Fin Plate':
1258                 continue
1259             if isinstance(chkbox, QCheckBox):
1260                 chkbox.setChecked(Qt.Unchecked)
1261                 ui.commLogicObj.display_3DModel("Plate", bgcolor)
```

## Appendix B

# Code for Save 3d models and cad images

```
2597     def save_cadImages(self,main):
2598         """Save CAD Model in image formats(PNG,JPEG,BMP,TIFF)"""
2599
2600         Returns:
2601
2602         """
2603
2604         if main.design_status:
2605
2606             files_types = "PNG (*.png);;JPEG (*.jpeg);;TIFF
2607             ↪ (*.tiff);;BMP(*.bmp)"
2608             fileName, _ = QFileDialog.getSaveFileName(self, 'Export',
2609             ↪ os.path.join(str(self.folder), "untitled.png"),
2610                                     files_types)
2611
2612             fName = str(fileName)
2613             file_extension = fName.split(".")[1]
2614
2615             if file_extension == 'png' or file_extension == 'jpeg' or
2616             ↪ file_extension == 'bmp' or file_extension == 'tiff':
2617                 self.display.ExportToImage(fName)
2618                 QMessageBox.about(self, 'Information', "File saved")
2619             else:
2620                 # self.actionSave_current_image.setEnabled(False)
2621                 QMessageBox.about(self, 'Information', 'Design Unsafe: CAD image
2622                 ↪ cannot be saved')
2623
2624     def save3DcadImages(self, main):
2625
2626         if not main.design_button_status:
2627             QMessageBox.warning(self, 'Warning', 'No design created!')
2628             return
2629
2630         if main.design_status:
2631             if self.fuse_model is None:
2632                 self.fuse_model = self.commLogicObj.create2Dcad()
2633             shape = self.fuse_model
```



```

2630     files_types = "IGS (*.igs);;STEP (*.stp);;STL
2631     ↪ (*.stl);;BREP(*.brep)"
2632
2632     fileName, _ = QFileDialog.getSaveFileName(self, 'Export',
2633     ↪ os.path.join(str(self.folder), "untitled.igs"),
2634     files_types)
2635
2635     fName = str(fileName)
2636
2636     if fName and self.fuse_model:
2637         file_extension = fName.split(".")[1]
2638
2639         if file_extension == 'igs':
2640             IGESControl.IGESControl_Controller().Init()
2641             iges_writer = IGESControl.IGESControl_Writer()
2642             iges_writer.AddShape(shape)
2643             iges_writer.Write(fName)
2644
2645         elif file_extension == 'brep':
2646
2647             BRepTools.breptools.Write(shape, fName)
2648
2649         elif file_extension == 'stp':
2650             # initialize the STEP exporter
2651             step_writer = STEPControl_Writer()
2652             Interface_Static_SetCVal("write.step.schema", "AP203")
2653
2654             # transfer shapes and write file
2655             step_writer.Transfer(shape, STEPControl_AsIs)
2656             status = step_writer.Write(fName)
2657
2658             assert (status == IFSelect_RetDone)
2659
2660         else:
2661             stl_writer = StlAPI_Writer()
2662             stl_writer.SetASCII Mode(True)
2663             stl_writer.Write(shape, fName)
2664
2665         self.fuse_model = None
2666
2667         QMessageBox.about(self, 'Information', "File saved")
2668
2669     else:
2670         QMessageBox.about(self, 'Error', "File not saved")
2671
2671     else:
2672         # self.actionSave_3D_model.setEnabled(False)
2673         QMessageBox.about(self, 'Warning', 'Design Unsafe: 3D Model cannot
2674         ↪ be saved')

```

## Appendix C

# Code for Custom Material Popup

```
2218
2219     def new_material_dialog(self):
2220         dialog = QtWidgets.QDialog(self)
2221         self.material_popup_message = ''
2222         self.invalid_field = ''
2223         dialog.setWindowTitle('Custom Material')
2224         layout = QtWidgets.QGridLayout(dialog)
2225         widget = QtWidgets.QWidget(dialog)
2226         widget.setLayout(layout)
2227         _translate = QtCore.QCoreApplication.translate
2228         textbox_list = ['Grade', 'Fy_20', 'Fy_20_40', 'Fy_40', 'Fu']
2229         event_function = [' ', self.material_popup_fy_20_event,
2230             ↪ self.material_popup_fy_20_40_event,
2231                 self.material_popup_fy_40_event,
2232             ↪ self.material_popup_fu_event]
2233         self.original_focus_event_functions = {}
2234
2235         i = 0
2236         for textbox_name in textbox_list:
2237             label = QtWidgets.QLabel(widget)
2238             label.setObjectName(textbox_name+"_label")
2239             font = QtGui.QFont()
2240             font.setPointSize(9)
2241             font.setBold(False)
2242             font.setWeight(50)
2243             label.setFont(font)
2244             label.setText(_translate("MainWindow", "<html><body><p>" +
2245                 ↪ textbox_name + "</p></body></html>"))
2246             # label.resize(120, 30)
2247             label.setFixedSize(100, 30)
2248             layout.addWidget(label, i, 1, 1, 1)
2249
2250             textbox = QtWidgets.QLineEdit(widget)
2251             textbox.setObjectName(textbox_name)
2252             font = QtGui.QFont()
2253             font.setPointSize(11)
2254             font.setBold(False)
2255             font.setWeight(50)
2256             textbox.setFont(font)
2257             # textbox.resize(120, 30)
```

```

2255     textbox.setFixedSize(200, 24)
2256     if textbox_name == 'Grade':
2257         textbox.setReadOnly(True)
2258         textbox.setText("Cus_----")
2259     else:
2260         textbox.setValidator(QtGui.QIntValidator())
2261         # textbox.mousePressEvent =
2262         ↪ event_function[textbox_list.index(textbox_name)]
2263         self.original_focus_event_functions.update({textbox_name:
2264         ↪ textbox.focusOutEvent})
2265         textbox.focusOutEvent =
2266         ↪ event_function[textbox_list.index(textbox_name)]
2267
2268     self.connect_change_popup_material(textbox, widget)
2269     layout.addWidget(textbox, i, 2, 1, 1)
2270
2271     i += 1
2272
2273     add_button = QtWidgets.QPushButton(widget)
2274     add_button.setObjectName("material_add")
2275     add_button.setText("Add")
2276     add_button.clicked.connect(lambda:
2277     ↪ self.update_material_db_validation(widget))
2278     layout.addWidget(add_button, i, 1, 1, 2)
2279
2280     dialog.setFixedSize(350, 250)
2281     closed = dialog.exec()
2282     if closed is not None:
2283         input_dock_material =
2284         ↪ self.dockWidgetContents.findChild(QtWidgets.QWidget,
2285         ↪ KEY_MATERIAL)
2286         input_dock_material.clear()
2287         for item in connectdb("Material"):
2288             input_dock_material.addItem(item)
2289         input_dock_material.setCurrentIndex(1)

```

## Appendix D

# Code for Download and Reset Database

```
970     def download_Database(self, table, call_type="database"):
971
972         fileName, _ = QFileDialog.getSaveFileName(QFileDialog(), "Download
↪ File", os.path.join(os.getcwd(), str(table+"_Details.xlsx")),
973                                                     "SectionDetails(*.xlsx)")
974
975         if not fileName:
976             return
977
978         try:
979             conn = sqlite3.connect(PATH_TO_DATABASE)
980             c = conn.cursor()
981             header = get_db_header(table)
982             wb = openpyxl.Workbook()
983             sheet = wb.create_sheet(table, 0)
984
985             col = 1
986             for head in header:
987                 sheet.cell(row=1, column=col).value = head
988                 col += 1
989
990             if call_type != "header":
991                 if table == 'Columns':
992                     c.execute("SELECT * FROM Columns")
993                 elif table == 'Beams':
994                     c.execute("SELECT * FROM Beams")
995                 elif table == 'Angles':
996                     c.execute("SELECT * FROM Angles")
997                 elif table == 'Channels':
998                     c.execute("SELECT * FROM Channels")
999
1000             data = c.fetchall()
1001             conn.commit()
1002             c.close()
1003             row = 2
1004             for rows in data:
1005                 col = 1
1006                 for cols in range(len(header)):
1007                     sheet.cell(row=row, column=col).value = rows[col - 1]
1008                     col += 1
1009                 row += 1
1010             wb.save(fileName)
```

```

1007     QMessageBox.information(QMessageBox(), 'Information', 'Your File is
        ↳ Downloaded.')
1008
1009     except IOError:
1010         QMessageBox.information(QMessageBox(), "Unable to save file",
1011                                 "There was an error saving \"%s\" " %
        ↳ fileName)
1012
1013     return

```

```

2834     def database_reset(self):
2835
2836         conn = sqlite3.connect(PATH_TO_DATABASE)
2837         tables = ["Columns", "Beams", "Angles", "Channels"]
2838         text = ""
2839         for table in tables:
2840             query = "DELETE FROM "+str(table)+" WHERE Source = ?"
2841             cursor = conn.execute(query, ('Custom',))
2842             text += str(table)+": "+str(cursor.rowcount)+" rows deleted. \n"
2843             conn.commit()
2844             cursor.close()
2845         conn.close()
2846         message = QMessageBox()
2847         message.setWindowTitle('Successful')
2848         message.addButton(message.Ok)
2849         message.setText(text)
2850         message.exec()

```

## Appendix E

# Code for Loading previous Inputs

```
1347
1348     last_design_folder = os.path.join('ResourceFiles', 'last_designs')
1349     last_design_file = str(main.module_name(main)).replace(' ', '') +
↪     ".osi"
1350     last_design_file = os.path.join(last_design_folder, last_design_file)
1351     last_design_dictionary = {}
1352     if not os.path.isdir(last_design_folder):
1353         os.mkdir(last_design_folder)
1354     if os.path.isfile(last_design_file):
1355         with open(str(last_design_file), 'r') as last_design:
1356             last_design_dictionary = yaml.safe_load(last_design)
1357     if isinstance(last_design_dictionary, dict):
1358         self.setDictToUserInputs(last_design_dictionary, option_list, data,
↪         new_list)
1359     if "out_titles_status" in last_design_dictionary.keys():
1360         title_status = last_design_dictionary["out_titles_status"]
1361         print("titles", title_status)
1362         title_count = 0
1363         out_titles = []
1364         title_repeat = 1
1365         for out_field in out_list:
1366             if out_field[2] == TYPE_TITLE:
1367                 title_name = out_field[1]
1368                 if title_name in out_titles:
1369                     title_name += str(title_repeat)
1370                     title_repeat += 1
1371                 if title_status[title_count] == 0:
1372                     self.output_title_fields[title_name][0].
1373                         setVisible(False)
1374                     title_count += 1
```

```
1931
1932     last_design_folder = os.path.join('ResourceFiles', 'last_designs')
1933     if not os.path.isdir(last_design_folder):
1934         os.mkdir(last_design_folder)
1935     last_design_file = str(main.module_name(main)).replace(' ', '') +
↪     ".osi"
1936     last_design_file = os.path.join(last_design_folder,
↪     last_design_file)
1937     out_titles_status = []
```

```
1938 out_titles = []
1939 title_repeat = 1
1940 for option in out_list:
1941     if option[2] == TYPE_TITLE:
1942         title_name = option[1]
1943         if title_name in out_titles:
1944             title_name += str(title_repeat)
1945             title_repeat += 1
1946         if self.output_title_fields[title_name][0].isVisible():
1947             out_titles_status.append(1)
1948         else:
1949             out_titles_status.append(0)
1950             out_titles.append(title_name)
1951 self.design_inputs.update({"out_titles_status": out_titles_status})
1952 with open(str(last_design_file), 'w') as last_design:
1953     yaml.dump(self.design_inputs, last_design)
1954 self.design_inputs.pop("out_titles_status")
```

## Appendix F

# Code for Help Options

```
1336     self.actionSample_Tutorials.triggered.connect(lambda:
1337     ↪ MyTutorials(self).exec())
1337     self.actionAbout_Osdag_2.triggered.connect(lambda:
1338     ↪ MyAboutOsdag(self).exec())
1338     self.actionAsk_Us_a_Question.triggered.connect(lambda:
1339     ↪ MyAskQuestion(self).exec())
1339     self.actionDesign_examples.triggered.connect(self.design_examples)
```

```
84
85 class MyTutorials(QDialog):
86     def __init__(self, parent=None):
87         QDialog.__init__(self, parent)
88         self.ui = Ui_Tutorial()
89         self.ui.setupUi(self)
90
91
92 class MyAboutOsdag(QDialog):
93     def __init__(self, parent=None):
94         QDialog.__init__(self, parent)
95         self.ui = Ui_AboutOsdag()
96         self.ui.setupUi(self)
97
98
99 class MyAskQuestion(QDialog):
100     def __init__(self, parent=None):
101         QDialog.__init__(self, parent)
102         self.ui = Ui_AskQuestion()
103         self.ui.setupUi(self)
104
```

```
299
300     def design_examples(self):
301         root_path = os.path.join('ResourceFiles', 'design_example', '_build',
302         ↪ 'html')
302         for html_file in os.listdir(root_path):
303             # if html_file.startswith('index'):
304             print(os.path.splitext(html_file)[1])
305             if os.path.splitext(html_file)[1] == '.html':
306                 if sys.platform == ("win32" or "win64"):
```



```
307         os.startfile(os.path.join(root_path, html_file))
308     else:
309         opener = "open" if sys.platform == "darwin" else "xdg-open"
310         subprocess.call([opener, "%s/%s" % (root_path, html_file)])
311
```

## Appendix G

# Code for Zoom-in, Zoom-out, Pan and Rotate options

```
1327     self.actionZoom_out.triggered.connect(lambda:
1328     ↪ self.display.ZoomFactor(1/1.1))
1329     self.actionZoom_in.triggered.connect(lambda:
1329     ↪ self.display.ZoomFactor(1.1))
1329     self.actionPan.triggered.connect(lambda:
1329     ↪ self.assign_display_mode(mode="pan"))
1330     self.actionRotate_3D_model.triggered.connect(lambda:
1330     ↪ self.assign_display_mode(mode="rotate"))
```

```
2572     key_function = {Qt.Key_Up: lambda: self.Pan_Rotate_model("Up"),
2573                    Qt.Key_Down: lambda: self.Pan_Rotate_model("Down"),
2574                    Qt.Key_Right: lambda: self.Pan_Rotate_model("Right"),
2575                    Qt.Key_Left: lambda: self.Pan_Rotate_model("Left")}
2576     self.modelTab._key_map.update(key_function)
```

```
2675     def assign_display_mode(self, mode):
2676
2677         self.modelTab.setFocus()
2678         if mode == "pan":
2679             self.display_mode = 'Pan'
2680         elif mode == "rotate":
2681             self.display_mode = 'Rotate'
2682         else:
2683             self.display_mode = 'Normal'
2684
2685     def Pan_Rotate_model(self, direction):
2686
2687         if self.display_mode == 'Pan':
2688             if direction == 'Up':
2689                 self.display.Pan(0, 10)
2690             elif direction == 'Down':
2691                 self.display.Pan(0, -10)
2692             elif direction == 'Left':
2693                 self.display.Pan(-10, 0)
2694             elif direction == 'Right':
2695                 self.display.Pan(10, 0)
2696         elif self.display_mode == 'Rotate':
```

```
2697     if direction == 'Up':
2698         self.display_y += 10
2699         self.display.Rotation(self.display_x, self.display_y)
2700     elif direction == 'Down':
2701         self.display_y -= 10
2702         self.display.Rotation(self.display_x, self.display_y)
2703     elif direction == 'Left':
2704         self.display_x -= 10
2705         self.display.Rotation(self.display_x, self.display_y)
2706     elif direction == 'Right':
2707         self.display_x += 10
2708         self.display.Rotation(self.display_x, self.display_y)
2709     else:
2710         pass
```

## Appendix H

# Code for Output Button Popup

```
2028     def output_button_dialog(self, main, button_list, button):
2029
2030         dialog = QtWidgets.QDialog()
2031         dialog.setObjectName("Dialog")
2032         layout1 = QtWidgets.QVBoxLayout(dialog)
2033
2034         note_widget = QWidget(dialog)
2035         note_layout = QVBoxLayout(note_widget)
2036         layout1.addWidget(note_widget)
2037
2038         scroll = QScrollArea(dialog)
2039         layout1.addWidget(scroll)
2040         scroll.setWidgetResizable(True)
2041         scroll.horizontalScrollBar().setVisible(False)
2042         scroll_content = QtWidgets.QWidget(scroll)
2043         outer_grid_layout = QtWidgets.QGridLayout(scroll_content)
2044         inner_grid_widget = QtWidgets.QWidget(scroll_content)
2045         image_widget = QtWidgets.QWidget(scroll_content)
2046         image_layout = QtWidgets.QVBoxLayout(image_widget)
2047         image_layout.setAlignment(Qt.AlignCenter)
2048         image_widget.setLayout(image_layout)
2049         inner_grid_layout = QtWidgets.QGridLayout(inner_grid_widget)
2050         inner_grid_widget.setLayout(inner_grid_layout)
2051         scroll_content.setLayout(outer_grid_layout)
2052         scroll.setWidget(scroll_content)
2053
2054         dialog_width = 260
2055         dialog_height = 300
2056         max_image_width = 0
2057         max_label_width = 0
2058         max_image_height = 0
2059
2060         section = 0
2061         no_note = True
2062
2063         for op in button_list:
2064
2065             if op[0] == button.objectName():
2066                 tup = op[3]
2067                 title = tup[0]
```

```

2068         fn = tup[1]
2069         dialog.setWindowTitle(title)
2070         j = 1
2071         _translate = QtCore.QCoreApplication.translate
2072         for option in fn(main, main.design_status):
2073             option_type = option[2]
2074             lable = option[1]
2075             value = option[3]
2076             if option_type in [TYPE_TEXTBOX, TYPE_COMBOBOX]:
2077                 l = QtWidgets.QLabel(inner_grid_widget)
2078                 font = QtGui.QFont()
2079                 font.setPointSize(9)
2080                 font.setBold(False)
2081                 font.setWeight(50)
2082                 l.setFont(font)
2083                 l.setObjectName(option[0] + "_label")
2084                 l.setText(_translate("MainWindow",
2085                                     ↪ "<html><head/><body><p>" + lable +
2086                                     ↪ "</p></body></html>"))
2087                 inner_grid_layout.addWidget(l, j, 1, 1, 1)
2088                 l.setFixedSize(l.sizeHint().width(),
2089                               ↪ l.sizeHint().height())
2090                 max_label_width = max(l.sizeHint().width(),
2091                                       ↪ max_label_width)
2092                 l.setSizePolicy(
2093                     QtWidgets.QSizePolicy(
2094                         QtWidgets.QSizePolicy.Maximum,
2095                         QtWidgets.QSizePolicy.Maximum))
2096
2097             if option_type == TYPE_SECTION:
2098                 if section != 0:
2099                     outer_grid_layout.addWidget(inner_grid_widget, j,
2100                                                 ↪ 1, 1, 1)
2101                     outer_grid_layout.addWidget(image_widget, j, 2, 1,
2102                                                 ↪ 1)
2103                     h11 = QtWidgets.QFrame()
2104                     h11.setFrameShape(QtWidgets.QFrame.HLine)
2105                     j += 1
2106                     outer_grid_layout.addWidget(h11, j, 1, 1, 2)
2107
2108             inner_grid_widget = QtWidgets.QWidget(scroll_content)
2109             image_widget = QtWidgets.QWidget(scroll_content)
2110             image_layout = QtWidgets.QVBoxLayout(image_widget)
2111             image_layout.setAlignment(Qt.AlignCenter)
2112             image_widget.setLayout(image_layout)
2113             inner_grid_layout =
2114             ↪ QtWidgets.QGridLayout(inner_grid_widget)
2115             inner_grid_widget.setLayout(inner_grid_layout)
2116             if value is not None and value != "":
2117                 im = QtWidgets.QLabel(image_widget)
2118                 im.setFixedSize(value[1], value[2])
2119                 pmap = QPixmap(value[0])
2120                 im.setScaledContents(1)
2121                 im.setPixmap(pmap)
2122                 image_layout.addWidget(im)
2123                 caption = QtWidgets.QLabel(image_widget)

```

```

2117         font = QtGui.QFont()
2118         font.setWeight(450)
2119         font.setPointSize(11)
2120         caption.setAlignment(Qt.AlignCenter)
2121         caption.setFont(font)
2122         caption.setText(value[3])
2123         caption.setFixedSize(value[1], 12)
2124         image_layout.addWidget(caption)
2125         max_image_width = max(max_image_width, value[1])
2126         max_image_height = max(max_image_height, value[2])
2127     j += 1
2128
2129     q = QtWidgets.QLabel(scroll_content)
2130     font = QtGui.QFont()
2131     font.setWeight(600)
2132     font.setPointSize(11)
2133     q.setFont(font)
2134     q.setObjectName("_title")
2135     q.setText(lable)
2136     q.setFixedSize(q.sizeHint().width(),
2137                  ↪ q.sizeHint().height())
2137     q.setSizePolicy(
2138         QtWidgets.QSizePolicy(
2139             QtWidgets.QSizePolicy.Maximum,
2140             QtWidgets.QSizePolicy.Maximum))
2141     outer_grid_layout.addWidget(q, j, 1, 1, 2)
2142     section += 1
2143
2144     if option_type == TYPE_TEXTBOX:
2145         r = QtWidgets.QLineEdit(inner_grid_widget)
2146         font = QtGui.QFont()
2147         font.setPointSize(11)
2148         font.setBold(False)
2149         font.setWeight(50)
2150         r.setFixedSize(160, 27)
2151         r.setFont(font)
2152         r.setObjectName(option[0])
2153         r.setText(str(value))
2154         inner_grid_layout.addWidget(r, j, 2, 1, 1)
2155
2156     if option_type == TYPE_IMAGE:
2157         im = QtWidgets.QLabel(image_widget)
2158         im.setScaledContents(True)
2159         im.setFixedSize(value[1], value[2])
2160         pmap = QPixmap(value[0])
2161         im.setPixmap(pmap)
2162         image_layout.addWidget(im)
2163         caption = QtWidgets.QLabel(image_widget)
2164         font = QtGui.QFont()
2165         font.setWeight(450)
2166         font.setPointSize(11)
2167         caption.setAlignment(Qt.AlignCenter)
2168         caption.setFont(font)
2169         caption.setText(value[3])
2170         caption.setFixedSize(value[1], 12)
2171         image_layout.addWidget(caption)

```

```

2172         max_image_width = max(max_image_width, value[1])
2173         max_image_height = max(max_image_height, value[2])
2174
2175         if option_type == TYPE_NOTE:
2176             note = QLabel(note_widget)
2177             font = QtGui.QFont()
2178             font.setWeight(450)
2179             font.setPointSize(11)
2180             note.setFont(font)
2181             note.setText("Note: "+str(value))
2182             note.setFixedSize(note.sizeHint().width(),
2183                               ↪ note.sizeHint().height())
2184             note_layout.addWidget(note)
2185             no_note = False
2186
2187         j = j + 1
2188
2189         if inner_grid_layout.count() > 0:
2190             outer_grid_layout.addWidget(inner_grid_widget, j, 1, 1, 1)
2191         if image_layout.count() > 0:
2192             outer_grid_layout.addWidget(image_widget, j, 2, 1, 1)
2193
2194         dialog_width += max_label_width
2195         dialog_width += max_image_width
2196         dialog_height = max(dialog_height, max_image_height)
2197         if not no_note:
2198             dialog_height += 40
2199         dialog.resize(dialog_width, dialog_height)
2200         dialog.setMinimumSize(dialog_width, dialog_height)
2201
2202         if no_note:
2203             layout1.removeWidget(note_widget)
2204
2205         dialog.exec()

```

## Appendix I

# Code for Browse, Save and Load profile

```
222     self.new_ui.btn_browse.clicked.connect(lambda:
223     ↪ self.getLogoFilePath(self.new_window, self.new_ui.lbl_browse))
224     self.new_ui.btn_saveProfile.clicked.connect(lambda:
225     ↪ self.saveUserProfile(self.new_window))
226     self.new_ui.btn_useProfile.clicked.connect(lambda:
227     ↪ self.useUserProfile(self.new_window))
```

```
229
230     def getLogoFilePath(self, window, lblwidget):
231
232         filename, _ = QFileDialog.getOpenFileName(window, "Open Image",
233         ↪ os.path.join(str(' '), ''), "InputFiles(*.png *.svg *.jpg)")
234
235         if filename == '':
236             return False
237         else:
238             lblwidget.setText(str(filename))
239
240         return str(filename)
```

```
255     def saveUserProfile(self, window):
256
257         inputData = self.getPopUpInputs()
258         filename, _ = QFileDialog.getSaveFileName(window, 'Save Files',
259         ↪ os.path.join(str(self.folder), "Profile"), '*.txt')
260
261         if filename == '':
262             return False
263         else:
264             infile = open(filename, 'w')
265             yaml.dump(inputData, infile)
266             infile.close()
```

```
283     def useUserProfile(self, window):
284         filename, _ = QFileDialog.getOpenFileName(
285         ↪ window, 'Open Files',
```



```
286         os.path.join(str(self.folder), "Profile"),
287         '*.txt')
288     if os.path.isfile(filename):
289         outfile = open(filename, 'r')
290         reportsummary = yaml.safe_load(outfile)
291         self.new_ui.lineEdit_companyName.setText(
292             reportsummary["ProfileSummary"]['CompanyName'])
293         self.new_ui.lbl_browse.setText(
294             reportsummary["ProfileSummary"]['CompanyLogo'])
295         self.new_ui.lineEdit_groupName.setText(
296             reportsummary["ProfileSummary"]['Group/TeamName'])
297         self.new_ui.lineEdit_designer.setText(
298             reportsummary["ProfileSummary"]['Designer'])
```

## Appendix J

# Code for Download and Import buttons

```
1020     def import_section(self, tab_name):
1021         fileName, _ = QFileDialog.getOpenFileName(QFileDialog(), "Open File",
1022             ↪ os.getcwd(),
1023                                                     "SectionDetails(*.xlsx)")
1024         if not fileName:
1025             return
1026         try:
1027             wb = openpyxl.load_workbook(fileName)
1028             if tab_name in wb.sheetnames:
1029                 if wb.sheetnames.count(tab_name) > 1:
1030                     QMessageBox.information(QMessageBox(), 'Information',
1031                                             str(' File contains multiple ' +
1032                                                 ↪ tab_name + ' Sheet.'))
1033                     return
1034
1035             sheet = wb[tab_name]
1036             header = []
1037             for cell in sheet[1]:
1038                 header.append(str(cell.value))
1039             if header == get_db_header(tab_name):
1040                 conn = sqlite3.connect(PATH_TO_DATABASE)
1041                 discarded = []
1042                 ignored = []
1043                 values = {}
1044                 for rows in range(2, sheet.max_row + 1):
1045                     for cols in range(1, len(header)+1):
1046                         key = header[cols - 1]
1047                         val = sheet.cell(row=rows, column=cols).value
1048                         if self.import_db_validation(tab_name, key, val):
1049                             values.update({key: val})
1050                         else:
1051                             discarded.append(sheet[rows][1].value)
1052                             break
1053                 c = conn.cursor()
1054                 if tab_name == 'Columns':
1055                     c.execute("SELECT count(*) FROM Columns WHERE
1056                                 ↪ Designation = ?", (values['Designation'],))
1057                 elif tab_name == 'Beams':
```

```

1055         c.execute("SELECT count(*) FROM Beams WHERE
1056             ↪ Designation = ?", (values['Designation'],))
1057     elif tab_name == 'Angles':
1058         c.execute("SELECT count(*) FROM Angles WHERE
1059             ↪ Designation = ?", (values['Designation'],))
1060     elif tab_name == 'Channels':
1061         c.execute("SELECT count(*) FROM Channels WHERE
1062             ↪ Designation = ?", (values['Designation'],))
1063
1064     data = c.fetchone()[0]
1065     if data == 0:
1066         values['Source'] = 'Custom'
1067         if tab_name == 'Columns':
1068             c.execute(''INSERT INTO Columns
1069                 ↪ (Designation,Mass,Area,D,B,tw,T,
1070                 ↪ FlangeSlope,R1,R2,Iz,Iy,rz,ry,Zz,Zy,Zpz,Zpy,It,Iw,Source,Type)
1071                 ↪ VALUES
1072                 ↪ (?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?)'',
1073                 ↪ (values['Designation'],
1074                 ↪ values['Mass'], values['Area'],
1075                 ↪ values['D'],
1076                 ↪ values['B'], values['tw'],
1077                 ↪ values['T'],
1078                 ↪ values['FlangeSlope'],
1079                 ↪ values['R1'], values['R2'],
1080                 ↪ values['Iz'], values['Iy'],
1081                 ↪ values['rz'],
1082                 ↪ values['ry'], values['Zz'],
1083                 ↪ values['Zy'], values['Zpz'],
1084                 ↪ values['Zpy'],
1085                 ↪ values['It'], values['Iw'],
1086                 ↪ values['Source'],
1087                 ↪ values['Type']))
1088         elif tab_name == 'Beams':
1089             c.execute(''INSERT INTO Beams
1090                 ↪ (Designation,Mass,Area,D,B,tw,T,
1091                 ↪ FlangeSlope,R1,R2,Iz,Iy,rz,ry,Zz,Zy,Zpz,Zpy,It,Iw,Source,Type)
1092                 ↪ VALUES(?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?,,?)'',
1093                 ↪ (values['Designation'],
1094                 ↪ values['Mass'], values['Area'],
1095                 ↪ values['D'],
1096                 ↪ values['B'], values['tw'],
1097                 ↪ values['T'],
1098                 ↪ values['FlangeSlope'],
1099                 ↪ values['R1'], values['R2'],
1100                 ↪ values['Iz'], values['Iy'],
1101                 ↪ values['rz'],
1102                 ↪ values['ry'], values['Zz'],
1103                 ↪ values['Zy'], values['Zpz'],
1104                 ↪ values['Zpy'],
1105                 ↪ values['It'], values['Iw'],
1106                 ↪ values['Source'],
1107                 ↪ values['Type']))

```

```

1082         elif tab_name == 'Angles':
1083             c.execute('''INSERT INTO Angles
1084                 ↪ (Designation,Mass,Area,a,b,t,R1,R2,
1085 ↪ Cz,Cy,Iz,Iy,Iumax,Ivmin,rz,ry,rumax,rvmin,Zz,Zy,Zpz,Zpy,It,Source,Type)
1086 ↪ VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?),
1087             (values['Designation'],
1088             ↪ values['Mass'], values['Area'],
1089             ↪ values['a'],
1090             values['b'], values['t'],
1091             ↪ values['R1'], values['R2'],
1092             ↪ values['Cz'],
1093             values['Cy'], values['Iz'],
1094             ↪ values['Iy'], values['Iumax'],
1095             ↪ values['Ivmin'],
1096             values['rz'], values['ry'],
1097             ↪ values['rumax'],
1098             ↪ values['rvmin'], values['Zz'],
1099             values['Zy'], values['Zpz'],
1100             ↪ values['Zpy'], values['It'],
1101             ↪ values['Source'],
1102             values['Type']))
1103         elif tab_name == 'Channels':
1104             c.execute('''INSERT INTO Channels
1105                 ↪ (Designation,Mass,Area,D,B,tw,T,
1106 ↪ FlangeSlope,R1,R2,Cy,Iz,Iy,rz,ry,Zz,Zy,Zpz,Zpy,Source,Type)
1107 ↪ VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?),
1108             (values['Designation'],
1109             ↪ values['Mass'], values['Area'],
1110             ↪ values['D'],
1111             values['B'], values['tw'],
1112             ↪ values['T'],
1113             ↪ values['FlangeSlope'],
1114             ↪ values['R1'],
1115             values['R2'], values['Cy'],
1116             ↪ values['Iz'], values['Iy'],
1117             ↪ values['rz'],
1118             values['ry'], values['Zz'],
1119             ↪ values['Zy'], values['Zpz'],
1120             ↪ values['Zpy'],
1121             values['Source'], values['Type']))
1122
1123         conn.commit()
1124         c.close()
1125
1126     else:
1127         ignored.append(values['Designation'])
1128
1129     conn.close()
1130     message = QMessageBox()
1131     message.setWindowTitle('Successful')
1132     message.addButton(message.Ok)

```

```

1112         message.setText('File data is imported successfully to the
↳ database.')
1113     if discarded or ignored:
1114         rejected = message.addButton('Rejected Sections',
↳ message.ActionRole)
1115         rejected.clicked.connect(lambda:
↳ self.import_validation_dialog(discarded, ignored))
1116     message.exec()
1117 else:
1118     QMessageBox.information(QMessageBox(), 'Information',
1119                             str(str(tab_name) + ' Sheet has
↳ headers different than
↳ database.))
1120
1121 else:
1122     QMessageBox.information(QMessageBox(), 'Information', str('
↳ File does not contain '+str(tab_name)+' Sheet.))
1123
1124 except IOError:
1125     QMessageBox.information(QMessageBox(), "Unable to open file",
1126                             "There was an error opening \"%s\" " %
↳ fileName)
1127
1128     return

```

## Appendix K

# Code for Import validation

```
1129     def import_db_validation(self, tab, key, value):
1130
1131         if key in ['Mass', 'Area', 'D', 'B', 'tw', 'T', 'FlangeSlope', 'R1',
1132                 ↪ 'R2',
1133                 'Iz', 'Iy', 'rz', 'ry', 'Zz', 'Zy', 'Zpz', 'Zpy', 'It',
1134                 ↪ 'Iw']:
1135             return isinstance(value, int) or isinstance(value, float)
1136         else:
1137             return True
1138
1139     def import_validation_dialog(self, discarded, ignored):
1140
1141         dialog = QDialog()
1142         dialog.setWindowTitle('Rejected Sections')
1143         vlayout = QVBoxLayout(dialog)
1144         height = 200
1145         total = len(discarded)+len(ignored)
1146         if 0 < total < 30:
1147             height += total*10
1148         else:
1149             height = 500
1150         dialog.resize(400, height)
1151         dialog.setLayout(vlayout)
1152         if discarded:
1153             scroll_discarded = QScrollArea(dialog)
1154             vlayout.addWidget(scroll_discarded)
1155             scroll_discarded.setWidgetResizable(True)
1156             scroll_discarded.setVerticalScrollBarPolicy(
1157                 QtCore.Qt.ScrollBarAsNeeded)
1158             widget_discarded = QWidget(scroll_discarded)
1159             layout_discarded = QVBoxLayout(widget_discarded)
1160             widget_discarded.setLayout(layout_discarded)
1161             label_discarded = QLabel("These values were rejected because of
1162                 ↪ validation.")
1163             layout_discarded.addWidget(label_discarded)
1164             scroll_discarded.setWidget(widget_discarded)
1165             text_discarded = QTextBrowser()
1166             layout_discarded.addWidget(text_discarded)
1167             for d in discarded:
1168                 text_discarded.append(d)
```

```
1166     if ignored:
1167         scroll_ignored = QScrollArea(dialog)
1168         vlayout.addWidget(scroll_ignored)
1169         scroll_ignored.setWidgetResizable(True)
1170         scroll_ignored.setVerticalScrollBarPolicy(
1171             QtCore.Qt.ScrollBarAsNeeded)
1172         widget_ignored = QWidget(scroll_ignored)
1173         layout_ignored = QVBoxLayout(widget_ignored)
1174         widget_ignored.setLayout(layout_ignored)
1175         label_ignored = QLabel("These values were ignored because they
1176             ↪ already exist in database.")
1177         layout_ignored.addWidget(label_ignored)
1178         scroll_ignored.setWidget(widget_ignored)
1179         text_ignored = QTextBrowser()
1180         layout_ignored.addWidget(text_ignored)
1181         for i in ignored:
1182             text_ignored.append(i)
1183     dialog.exec()
```

## Appendix L

# Code for New features

```
230     def change_source(self):
231
232         designation = self[0]
233         source = 'Custom'
234         if designation in connectdb("Columns", call_type="dropdown"):
235             source = get_source("Columns", designation)
236         elif designation in connectdb("Beams", call_type="dropdown"):
237             source = get_source("Beams", designation)
238         elif designation in connectdb("Angles", call_type="dropdown"):
239             source = get_source("Angles", designation)
240         elif designation in connectdb("Channels", call_type="dropdown"):
241             source = get_source("Channels", designation)
242
243         d = {'Label_23': str(source),
244             'Label_24': str(source),
245             'Label_21': str(source)}
246         return d
```

```
214     def get_source(table_name, designation):
215
216         conn = sqlite3.connect(PATH_TO_DATABASE)
217
218         if table_name == "Angles":
219             cursor = conn.execute("SELECT Source FROM Angles WHERE Designation =
220             ↪ ?", (designation,))
221
222         elif table_name == "Channels":
223             cursor = conn.execute("SELECT Source FROM Channels WHERE Designation =
224             ↪ ?", (designation,))
225
226         elif table_name == "Beams":
227             cursor = conn.execute("SELECT Source FROM Beams WHERE Designation = ?",
228             ↪ (designation,))
229
230         else:
231             cursor = conn.execute("SELECT Source FROM Columns WHERE Designation =
232             ↪ ?", (designation,))
233
234         source = cursor.fetchone()[0]
235         return str(source)
```



```

278     if lable in [KEY_DISP_FU, KEY_DISP_FY, KEY_DISP_POISSON_RATIO,
279 ↪ KEY_DISP_THERMAL_EXP,
280     KEY_DISP_MOD_OF_ELAST, KEY_DISP_MOD_OF_RIGID, 'Source']:
281         line.setReadOnly(True)
282         self.do_not_clear_list.append(line)
283     if main.module_name(main) in [KEY_DISP_TENSION_BOLTED,
284 ↪ KEY_DISP_TENSION_WELDED] and lable in \
285     [KEY_DISP_LOCATION, KEY_DISP_SEC_PROFILE]:
286         line.setReadOnly(True)
287         self.do_not_clear_list.append(line)
288     if last_title == KEY_DISP_DIMENSIONS:
289         if element[1] in [KEY_DISP_ROOT_R, KEY_DISP_TOE_R]:
290             regex_validator = QtCore.QRegExp("[0-9]*[.][0-9]*|.[0-9]*|0")
291         else:
292             regex_validator = QtCore.QRegExp("[1-9][0-9]*[.][0-9]*|.[0-9]*")
293         line.setValidator(QtGui.QRegExpValidator(regex_validator, line))
294     if last_title == KEY_DISP_SEC_PROP:
295         regex_validator =
296 ↪ QtCore.QRegExp("[1-9][0-9]*[.][0-9]*|.[0-9]*|N/A|-")
297         line.setValidator(QtGui.QRegExpValidator(regex_validator, line))

```