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

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 changig 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.

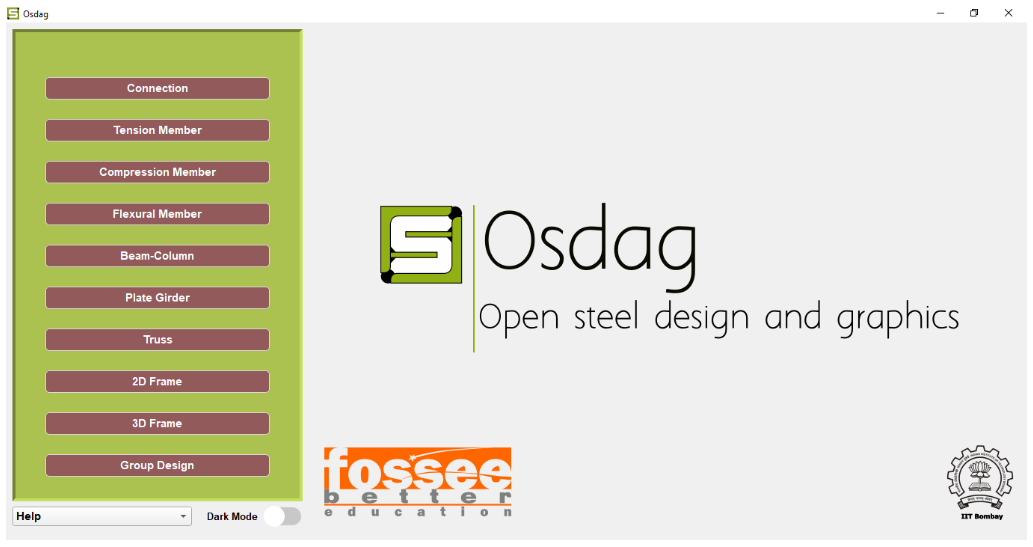


Figure 2.1: Osdag Mainpage

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. Thes checkboxes are used to show different components of the design in 3d viewer. Concerned code is attached vide Appendix-A.

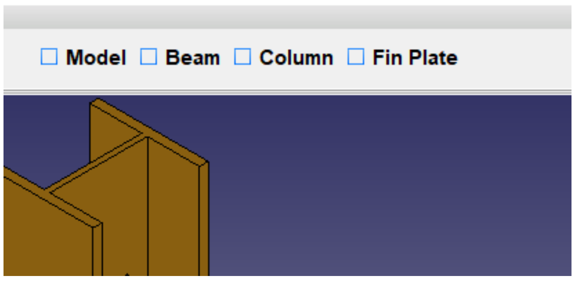


Figure 2.2: Checkbox

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

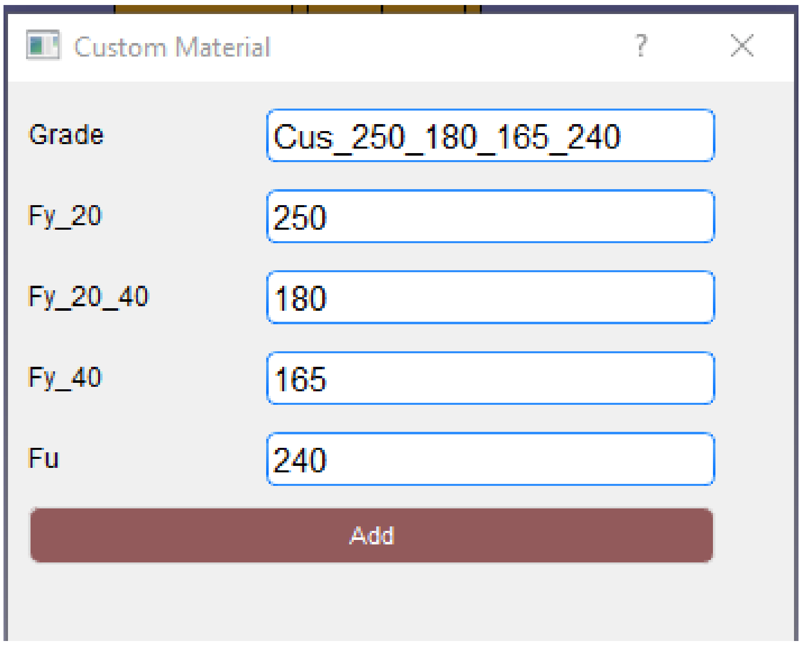
This popup allows user to add customized matrial grdae in database and use that material grade for design. Concerned code is attached vide Appendix-C.

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-


Custom Material

Grade	Cus_250_180_165_240
Fy_20	250
Fy_20_40	180
Fy_40	165
Fu	240

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.

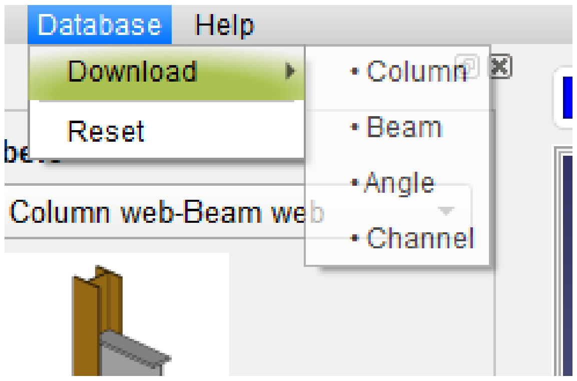


Figure 2.4: Download and Reset Database

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.

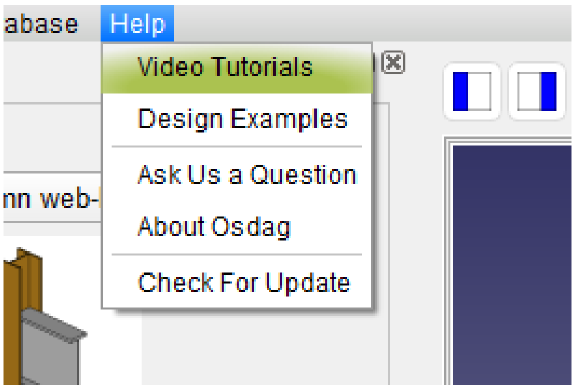


Figure 2.5: Help

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.

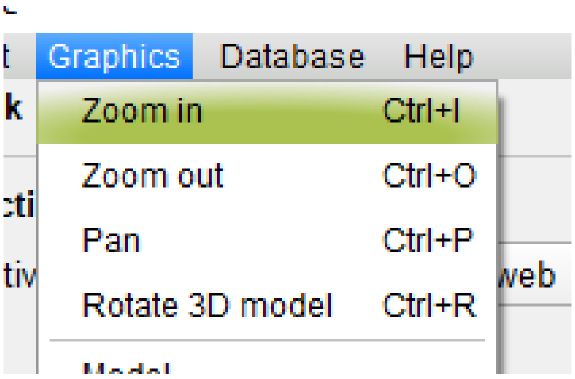
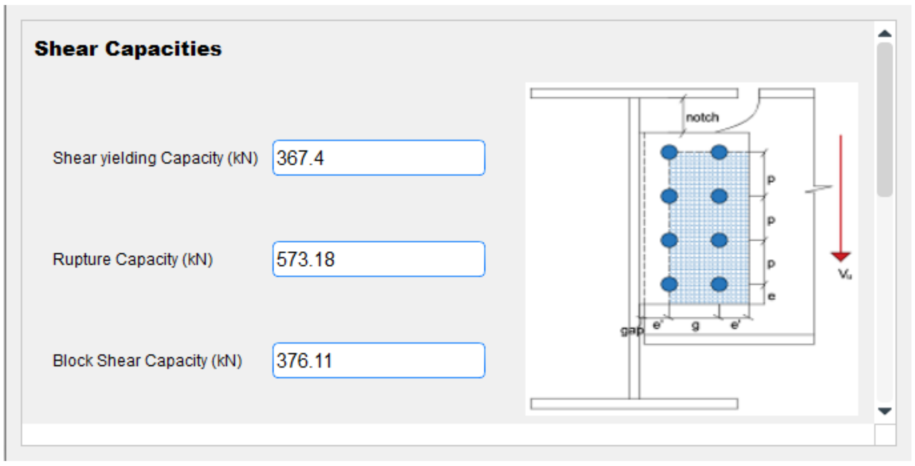


Figure 2.6: Zoom-in Zoom-out Pan and Rotate

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.



Shear Capacities

Shear yielding Capacity (kN) 367.4

Rupture Capacity (kN) 573.18

Block Shear Capacity (kN) 376.11

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.

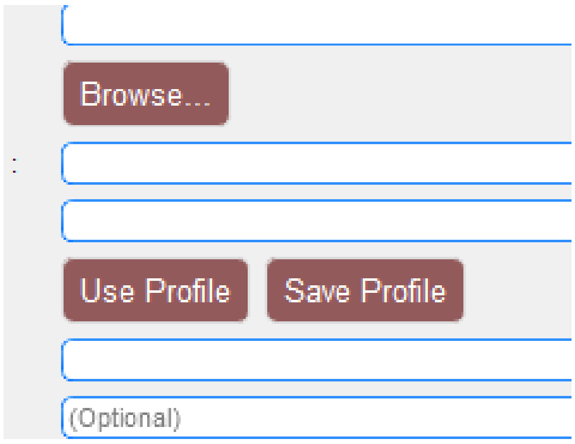


Figure 2.8: Browse, Save and Load profile

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.

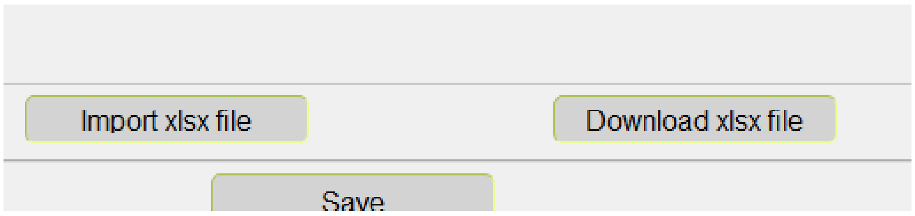


Figure 2.9: Download and Import

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

```

    ########################################
    # Function for individual component calls in 3D view
    ########################################
    def get_3d_components(self):
        components = []

        t1 = ('Model', self.call_3DModel)
        components.append(t1)

        t2 = ('Beam', self.call_3DBeam)
        components.append(t2)

        t3 = ('Column', self.call_3DColumn)
        components.append(t3)

        t4 = ('Fin Plate', self.call_3DPlate)
        components.append(t4)

        return components

    def call_3DPlate(self, ui, bgcolor):
        from PyQt5.QtWidgets import QCheckBox
        from PyQt5.QtCore import Qt
        for chkbox in ui.frame.children():
            if chkbox.objectName() == 'Fin Plate':
                continue
            if isinstance(chkbox, QCheckBox):
                chkbox.setChecked(Qt.Unchecked)
        ui.commLogicObj.display_3DModel("Plate", bgcolor)
```

Appendix B

Code for Save 3d models and cad images

```
    def save_cadImages(self,main):
        """Save CAD Model in image formats(PNG,JPEG,BMP,TIFF)

        Returns:

        """

        if main.design_status:

            files_types = "PNG (*.png);;JPEG (*.jpeg);;TIFF
            ↪   (*.tiff);;BMP(*.bmp)"
            fileName, _ = QFileDialog.getSaveFileName(self, 'Export',
            ↪   os.path.join(str(self.folder), "untitled.png"),
                                                    files_types)
            fName = str(fileName)
            file_extension = fName.split(".")[-1]

            if file_extension == 'png' or file_extension == 'jpeg' or
            ↪   file_extension == 'bmp' or file_extension == 'tiff':
                self.display.ExportToImage(fName)
                QMessageBox.about(self, 'Information', "File saved")
        else:
            # self.actionSave_current_image.setEnabled(False)
            QMessageBox.about(self, 'Information', 'Design Unsafe: CAD image
            ↪   cannot be saved')

    def save3DcadImages(self, main):

        if not main.design_button_status:
            QMessageBox.warning(self, 'Warning', 'No design created!')
            return

        if main.design_status:
            if self.fuse_model is None:
                self.fuse_model = self.commLogicObj.create2Dcad()
            shape = self.fuse_model

            files_types = "IGS (*.igs);;STEP (*.stp);;STL
            ↪  (*.stl);;BREP(*.brep)"

            fileName, _ = QFileDialog.getSaveFileName(self, 'Export',
            ↪  os.path.join(str(self.folder), "untitled.igs"),
                                                      files_types)
            fName = str(fileName)

            if fName and self.fuse_model:
                file_extension = fName.split(".")[-1]

                if file_extension == 'igs':
                    IGESControl.IGESControl_Controller().Init()
                    iges_writer = IGESControl.IGESControl_Writer()
                    iges_writer.AddShape(shape)
                    iges_writer.Write(fName)

                elif file_extension == 'brep':

                    BRepTools.breptools.Write(shape, fName)

                elif file_extension == 'stp':
                    # initialize the STEP exporter
                    step_writer = STEPControl_Writer()
                    Interface_Static_SetCVal("write.step.schema", "AP203")

                    # transfer shapes and write file
                    step_writer.Transfer(shape, STEPControl_AsIs)
                    status = step_writer.Write(fName)

                    assert (status == IFSelect_RetDone)

                else:
                    stl_writer = StlAPI_Writer()
                    stl_writer.SetASCIIMode(True)
                    stl_writer.Write(shape, fName)

                self.fuse_model = None

                QMessageBox.about(self, 'Information', "File saved")

            else:
                QMessageBox.about(self, 'Error', "File not saved")
        else:
            # self.actionSave_3D_model.setEnabled(False)
            QMessageBox.about(self, 'Warning', 'Design Unsafe: 3D Model cannot
            ↪  be saved')

```

Appendix C

Code for Custom Material Popup

```

    def new_material_dialog(self):
        dialog = QtWidgets.QDialog(self)
        self.material_popup_message = ''
        self.invalid_field = ''
        dialog.setWindowTitle('Custom Material')
        layout = QtWidgets.QGridLayout(dialog)
        widget = QtWidgets.QWidget(dialog)
        widget.setLayout(layout)
        _translate = QtCore.QCoreApplication.translate
        textbox_list = ['Grade', 'Fy_20', 'Fy_20_40', 'Fy_40', 'Fu']
        event_function = ['', self.material_popup_fy_20_event,
        ↪  self.material_popup_fy_20_40_event,
                              self.material_popup_fy_40_event,
                          ↪  self.material_popup_fu_event]
        self.original_focus_event_functions = {}

        i = 0
        for textbox_name in textbox_list:
            label = QtWidgets.QLabel(widget)
            label.setObjectName(textbox_name+"_label")
            font = QtGui.QFont()
            font.setPointSize(9)
            font.setBold(False)
            font.setWeight(50)
            label.setFont(font)
            label.setText(_translate("MainWindow", "<html><body><p>" +
            ↪  textbox_name + "</p></body></html>"))
            # label.resize(120, 30)
            label.setFixedSize(100, 30)
            layout.addWidget(label, i, 1, 1, 1)

            textbox = QtWidgets.QLineEdit(widget)
            textbox.setObjectName(textbox_name)
            font = QtGui.QFont()
            font.setPointSize(11)
            font.setBold(False)
            font.setWeight(50)
            textbox.setFont(font)
            # textbox.resize(120, 30)
            textbox.setFixedSize(200, 24)
            if textbox_name == 'Grade':
                textbox.setReadOnly(True)
                textbox.setText("Cus____")
            else:
                textbox.setValidator(QtGui.QIntValidator())
                # textbox.mousePressEvent =
                ↪   event_function[textbox_list.index(textbox_name)]
                self.original_focus_event_functions.update({textbox_name:
                ↪   textbox.focusOutEvent})
                textbox.focusOutEvent =
                ↪   event_function[textbox_list.index(textbox_name)]

            self.connect_change_popup_material(textbox, widget)
            layout.addWidget(textbox, i, 2, 1, 1)

            i += 1

        add_button = QtWidgets.QPushButton(widget)
        add_button.setObjectName("material_add")
        add_button.setText("Add")
        add_button.clicked.connect(lambda:
        ↪   self.update_material_db_validation(widget))
        layout.addWidget(add_button, i, 1, 1, 2)

        dialog.setFixedSize(350, 250)
        closed = dialog.exec()
        if closed is not None:
            input_dock_material =
            ↪   self.dockWidgetContents.findChild(QtWidgets.QWidget,
            ↪   KEY_MATERIAL)
            input_dock_material.clear()
            for item in connectdb("Material"):
                input_dock_material.addItem(item)
            input_dock_material.setCurrentIndex(1)

```

Appendix D

Code for Download and Reset Database

```
    def download_Database(self, table, call_type="database"):

        fileName, _ = QFileDialog.getSaveFileName(QFileDialog(), "Download
        ↪  File", os.path.join(os.getcwd(), str(table+"_Details.xlsx")),
                                                      "SectionDetails(*.xlsx)")
        if not fileName:
            return
        try:
            conn = sqlite3.connect(PATH_TO_DATABASE)
            c = conn.cursor()
            header = get_db_header(table)
            wb = openpyxl.Workbook()
            sheet = wb.create_sheet(table, 0)

            col = 1
            for head in header:
                sheet.cell(row=1, column=col).value = head
                col += 1
            if call_type != "header":
                if table == 'Columns':
                    c.execute("SELECT * FROM Columns")
                elif table == 'Beams':
                    c.execute("SELECT * FROM Beams")
                elif table == 'Angles':
                    c.execute("SELECT * FROM Angles")
                elif table == 'Channels':
                    c.execute("SELECT * FROM Channels")
                data = c.fetchall()
                conn.commit()
                c.close()
                row = 2
                for rows in data:
                    col = 1
                    for cols in range(len(header)):
                        sheet.cell(row=row, column=col).value = rows[col - 1]
                        col += 1
                    row += 1
            wb.save(fileName)
            QMessageBox.information(QMessageBox(), 'Information', 'Your File is
            ↪  Downloaded.')

        except IOError:
            QMessageBox.information(QMessageBox(), "Unable to save file",
                                        "There was an error saving \"%s\"" %
                                        ↪  fileName)
            return

```

```
    def database_reset(self):

        conn = sqlite3.connect(PATH_TO_DATABASE)
        tables = ["Columns", "Beams", "Angles", "Channels"]
        text = ""
        for table in tables:
            query = "DELETE FROM "+str(table)+" WHERE Source = ?"
            cursor = conn.execute(query, ('Custom',))
            text += str(table)+": "+str(cursor.rowcount)+" rows deleted. \n"
            conn.commit()
            cursor.close()
        conn.close()
        message = QMessageBox()
        message.setWindowTitle('Successful')
        message.addButton(message.Ok)
        message.setText(text)
        message.exec()
```

Appendix E

Code for Loading previous Inputs

```

        last_design_folder = os.path.join('ResourceFiles', 'last_designs')
        last_design_file = str(main.module_name(main)).replace(' ', '') +
        ↪  ".osi"
        last_design_file = os.path.join(last_design_folder, last_design_file)
        last_design_dictionary = {}
        if not os.path.isdir(last_design_folder):
            os.mkdir(last_design_folder)
        if os.path.isfile(last_design_file):
            with open(str(last_design_file), 'r') as last_design:
                last_design_dictionary = yaml.safe_load(last_design)
        if isinstance(last_design_dictionary, dict):
            self.setDictToUserInputs(last_design_dictionary, option_list, data,
            ↪  new_list)
            if "out_titles_status" in last_design_dictionary.keys():
                title_status = last_design_dictionary["out_titles_status"]
                print("titles", title_status)
                title_count = 0
                out_titles = []
                title_repeat = 1
                for out_field in out_list:
                    if out_field[2] == TYPE_TITLE:
                        title_name = out_field[1]
                        if title_name in out_titles:
                            title_name += str(title_repeat)
                            title_repeat += 1
                        if title_status[title_count] == 0:
                            self.output_title_fields[title_name][0].
                            setVisible(False)
                        title_count += 1
```

```

            last_design_folder = os.path.join('ResourceFiles', 'last_designs')
            if not os.path.isdir(last_design_folder):
                os.mkdir(last_design_folder)
            last_design_file = str(main.module_name(main)).replace(' ', '') +
            ↪  ".osi"
            last_design_file = os.path.join(last_design_folder,
            ↪  last_design_file)
            out_titles_status = []
            out_titles = []
            title_repeat = 1
            for option in out_list:
                if option[2] == TYPE_TITLE:
                    title_name = option[1]
                    if title_name in out_titles:
                        title_name += str(title_repeat)
                        title_repeat += 1
                    if self.output_title_fields[title_name][0].isVisible():
                        out_titles_status.append(1)
                    else:
                        out_titles_status.append(0)
                    out_titles.append(title_name)
            self.design_inputs.update({"out_titles_status": out_titles_status})
            with open(str(last_design_file), 'w') as last_design:
                yaml.dump(self.design_inputs, last_design)
            self.design_inputs.pop("out_titles_status")
```

Appendix F

Code for Help Options

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

```

class MyTutorials(QDialog):
    def __init__(self, parent=None):
        QDialog.__init__(self, parent)
        self.ui = Ui_Tutorial()
        self.ui.setupUi(self)


class MyAboutOsdag(QDialog):
    def __init__(self, parent=None):
        QDialog.__init__(self, parent)
        self.ui = Ui_AboutOsdag()
        self.ui.setupUi(self)


class MyAskQuestion(QDialog):
    def __init__(self, parent=None):
        QDialog.__init__(self, parent)
        self.ui = Ui_AskQuestion()
        self.ui.setupUi(self)

```

```

    def design_examples(self):
        root_path = os.path.join('ResourceFiles', 'design_example', '_build',
        ↪  'html')
        for html_file in os.listdir(root_path):
            # if html_file.startswith('index'):
            print(os.path.splitext(html_file)[1])
            if os.path.splitext(html_file)[1] == '.html':
                if sys.platform == ("win32" or "win64"):
                    os.startfile(os.path.join(root_path, html_file))
                else:
                    opener ="open" if sys.platform == "darwin" else "xdg-open"
                    subprocess.call([opener, "%s/%s" % (root_path, html_file)])

```

Appendix G

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

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

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

```python
    def assign_display_mode(self, mode):

        self.modelTab.setFocus()
        if mode == "pan":
            self.display_mode = 'Pan'
        elif mode == "rotate":
            self.display_mode = 'Rotate'
        else:
            self.display_mode = 'Normal'

    def Pan_Rotate_model(self, direction):

        if self.display_mode == 'Pan':
            if direction == 'Up':
                self.display.Pan(0, 10)
            elif direction == 'Down':
                self.display.Pan(0, -10)
            elif direction == 'Left':
                self.display.Pan(-10, 0)
            elif direction == 'Right':
                self.display.Pan(10, 0)
        elif self.display_mode == 'Rotate':
            if direction == 'Up':
                self.display_y += 10
                self.display.Rotation(self.display_x, self.display_y)
            elif direction == 'Down':
                self.display_y -= 10
                self.display.Rotation(self.display_x, self.display_y)
            elif direction == 'Left':
                self.display_x -= 10
                self.display.Rotation(self.display_x, self.display_y)
            elif direction == 'Right':
                self.display_x += 10
                self.display.Rotation(self.display_x, self.display_y)
        else:
            pass
```

Appendix H

Code for Output Button Popup

```
    def output_button_dialog(self, main, button_list, button):

        dialog = QtWidgets.QDialog()
        dialog.setObjectName("Dialog")
        layout1 = QtWidgets.QVBoxLayout(dialog)

        note_widget = QWidget(dialog)
        note_layout = QVBoxLayout(note_widget)
        layout1.addWidget(note_widget)

        scroll = QScrollArea(dialog)
        layout1.addWidget(scroll)
        scroll.setWidgetResizable(True)
        scroll.horizontalScrollBar().setVisible(False)
        scroll_content = QtWidgets.QWidget(scroll)
        outer_grid_layout = QtWidgets.QGridLayout(scroll_content)
        inner_grid_widget = QtWidgets.QWidget(scroll_content)
        image_widget = QtWidgets.QWidget(scroll_content)
        image_layout = QtWidgets.QVBoxLayout(image_widget)
        image_layout.setAlignment(Qt.AlignCenter)
        image_widget.setLayout(image_layout)
        inner_grid_layout = QtWidgets.QGridLayout(inner_grid_widget)
        inner_grid_widget.setLayout(inner_grid_layout)
        scroll_content.setLayout(outer_grid_layout)
        scroll.setWidget(scroll_content)

        dialog_width = 260
        dialog_height = 300
        max_image_width = 0
        max_label_width = 0
        max_image_height = 0

        section = 0
        no_note = True

        for op in button_list:

            if op[0] == button.objectName():
                tup = op[3]
                title = tup[0]
                fn = tup[1]
                dialog.setWindowTitle(title)
                j = 1
                _translate = QtCore.QCoreApplication.translate
                for option in fn(main, main.design_status):
                    option_type = option[2]
                    lable = option[1]
                    value = option[3]
                    if option_type in [TYPE_TEXTBOX, TYPE_COMBOBOX]:
                        l = QtWidgets.QLabel(inner_grid_widget)
                        font = QtGui.QFont()
                        font.setPointSize(9)
                        font.setBold(False)
                        font.setWeight(50)
                        l.setFont(font)
                        l.setObjectName(option[0] + "_label")
                        l.setText(_translate("MainWindow",
                        ↪   "<html><head/><body><p>" + lable +
                        ↪   "</p></body></html>"))
                        inner_grid_layout.addWidget(l, j, 1, 1, 1)
                        l.setFixedSize(l.sizeHint().width(),
                        ↪   l.sizeHint().height())
                        max_label_width = max(l.sizeHint().width(),
                        ↪   max_label_width)
                        l.setSizePolicy(
                            QtWidgets.QSizePolicy(
                                QtWidgets.QSizePolicy.Maximum,
                                QtWidgets.QSizePolicy.Maximum))

                    if option_type == TYPE_SECTION:
                        if section != 0:
                            outer_grid_layout.addWidget(inner_grid_widget, j,
                            ↪   1, 1, 1)
                            outer_grid_layout.addWidget(image_widget, j, 2, 1,
                            ↪   1)
                            hl1 = QtWidgets.QFrame()
                            hl1.setFrameShape(QtWidgets.QFrame.HLine)
                            j += 1
                            outer_grid_layout.addWidget(hl1, j, 1, 1, 2)

                        inner_grid_widget = QtWidgets.QWidget(scroll_content)
                        image_widget = QtWidgets.QWidget(scroll_content)
                        image_layout = QtWidgets.QVBoxLayout(image_widget)
                        image_layout.setAlignment(Qt.AlignCenter)
                        image_widget.setLayout(image_layout)
                        inner_grid_layout =
                        ↪   QtWidgets.QGridLayout(inner_grid_widget)
                        inner_grid_widget.setLayout(inner_grid_layout)
                        if value is not None and value != "":
                            im = QtWidgets.QLabel(image_widget)
                            im.setFixedSize(value[1], value[2])
                            pmap = QPixmap(value[0])
                            im.setScaledContents(1)
                            im.setPixmap(pmap)
                            image_layout.addWidget(im)
                            caption = QtWidgets.QLabel(image_widget)
                            font = QtGui.QFont()
                            font.setWeight(450)
                            font.setPointSize(11)
                            caption.setAlignment(Qt.AlignCenter)
                            caption.setFont(font)
                            caption.setText(value[3])
                            caption.setFixedSize(value[1], 12)
                            image_layout.addWidget(caption)
                            max_image_width = max(max_image_width, value[1])
                            max_image_height = max(max_image_height, value[2])
                        j += 1

                        q = QtWidgets.QLabel(scroll_content)
                        font = QtGui.QFont()
                        font.setWeight(600)
                        font.setPointSize(11)
                        q.setFont(font)
                        q.setObjectName("_title")
                        q.setText(lable)
                        q.setFixedSize(q.sizeHint().width(),
                        ↪   q.sizeHint().height())
                        q.setSizePolicy(
                            QtWidgets.QSizePolicy(
                                QtWidgets.QSizePolicy.Maximum,
                                QtWidgets.QSizePolicy.Maximum))
                        outer_grid_layout.addWidget(q, j, 1, 1, 2)
                        section += 1

                    if option_type == TYPE_TEXTBOX:
                        r = QtWidgets.QLineEdit(inner_grid_widget)
                        font = QtGui.QFont()
                        font.setPointSize(11)
                        font.setBold(False)
                        font.setWeight(50)
                        r.setFixedSize(160, 27)
                        r.setFont(font)
                        r.setObjectName(option[0])
                        r.setText(str(value))
                        inner_grid_layout.addWidget(r, j, 2, 1, 1)

                    if option_type == TYPE_IMAGE:
                        im = QtWidgets.QLabel(image_widget)
                        im.setScaledContents(True)
                        im.setFixedSize(value[1], value[2])
                        pmap = QPixmap(value[0])
                        im.setPixmap(pmap)
                        image_layout.addWidget(im)
                        caption = QtWidgets.QLabel(image_widget)
                        font = QtGui.QFont()
                        font.setWeight(450)
                        font.setPointSize(11)
                        caption.setAlignment(Qt.AlignCenter)
                        caption.setFont(font)
                        caption.setText(value[3])
                        caption.setFixedSize(value[1], 12)
                        image_layout.addWidget(caption)
                        max_image_width = max(max_image_width, value[1])
                        max_image_height = max(max_image_height, value[2])

                    if option_type == TYPE_NOTE:
                        note = QLabel(note_widget)
                        font = QtGui.QFont()
                        font.setWeight(450)
                        font.setPointSize(11)
                        note.setFont(font)
                        note.setText("Note: "+str(value))
                        note.setFixedSize(note.sizeHint().width(),
                        ↪  note.sizeHint().height())
                        note_layout.addWidget(note)
                        no_note = False

                    j = j + 1

                if inner_grid_layout.count() > 0:
                    outer_grid_layout.addWidget(inner_grid_widget, j, 1, 1, 1)
                if image_layout.count() > 0:
                    outer_grid_layout.addWidget(image_widget, j, 2, 1, 1)

                dialog_width += max_label_width
                dialog_width += max_image_width
                dialog_height = max(dialog_height, max_image_height)
                if not no_note:
                    dialog_height += 40
                dialog.resize(dialog_width, dialog_height)
                dialog.setMinimumSize(dialog_width, dialog_height)

                if no_note:
                    layout1.removeWidget(note_widget)

                dialog.exec()
```

Appendix I

Code for Browse, Save and Load profile

```python
        self.new_ui.btn_browse.clicked.connect(lambda: self.getLogoFilePath(self.new_window, self.new_ui.lbl_browse))
        self.new_ui.btn_saveProfile.clicked.connect(lambda: self.saveUserProfile(self.new_window))
        self.new_ui.btn_useProfile.clicked.connect(lambda: self.useUserProfile(self.new_window))
```

```python

    def getLogoFilePath(self, window, lblwidget):

        filename, _ = QFileDialog.getOpenFileName(window, "Open Image", os.path.join(str(' '), ''), "InputFiles(*.png *.svg *.jpg)")

        if filename == '':
            return False
        else:
            lblwidget.setText(str(filename))

        return str(filename)
```

```python
    def saveUserProfile(self, window):

        inputData = self.getPopUpInputs()
        filename, _ = QFileDialog.getSaveFileName(window, 'Save Files',
                                os.path.join(str(self.folder), "Profile"), '*.txt')
        if filename == '':
            return False
        else:
            infile = open(filename, 'w')
            yaml.dump(inputData, infile)
            infile.close()

```

```python
    def useUserProfile(self, window):
        filename, _ = QFileDialog.getOpenFileName(
            window, 'Open Files',
            os.path.join(str(self.folder), "Profile"),
            '*.txt')
        if os.path.isfile(filename):
            outfile = open(filename, 'r')
            reportsummary = yaml.safe_load(outfile)
            self.new_ui.lineEdit_companyName.setText(
                reportsummary["ProfileSummary"]['CompanyName'])
            self.new_ui.lbl_browse.setText(
                reportsummary["ProfileSummary"]['CompanyLogo'])
            self.new_ui.lineEdit_groupName.setText(
                reportsummary["ProfileSummary"]['Group/TeamName'])
            self.new_ui.lineEdit_designer.setText(
                reportsummary["ProfileSummary"]['Designer'])
```

Appendix J

Code for Download and Import buttons

```
    def import_section(self, tab_name):
        fileName, _ = QFileDialog.getOpenFileName(QFileDialog(), "Open File",
        ↪  os.getcwd(),
                                                "SectionDetails(*.xlsx)")
        if not fileName:
            return
        try:
            wb = openpyxl.load_workbook(fileName)
            if tab_name in wb.sheetnames:
                if wb.sheetnames.count(tab_name) > 1:
                    QMessageBox.information(QMessageBox(), 'Information',
                                            str(' File contains multiple ' +
                                            ↪  tab_name + ' Sheet.'))
                    return

                sheet = wb[tab_name]
                header = []
                for cell in sheet[1]:
                    header.append(str(cell.value))
                if header == get_db_header(tab_name):
                    conn = sqlite3.connect(PATH_TO_DATABASE)
                    discarded = []
                    ignored = []
                    values = {}
                    for rows in range(2, sheet.max_row + 1):
                        for cols in range(1, len(header)+1):
                            key = header[cols - 1]
                            val = sheet.cell(row=rows, column=cols).value
                            if self.import_db_validation(tab_name, key, val):
                                values.update({key: val})
                            else:
                                discarded.append(sheet[rows][1].value)
                                break
                        c = conn.cursor()
                        if tab_name == 'Columns':
                            c.execute("SELECT count(*) FROM Columns WHERE
                            ↪  Designation = ?", (values['Designation'],))
                        elif tab_name == 'Beams':
```

```
                    c.execute("SELECT count(*) FROM Beams WHERE Designation = ?", (values['Designation'],))
                elif tab_name == 'Angles':
                    c.execute("SELECT count(*) FROM Angles WHERE Designation = ?", (values['Designation'],))
                elif tab_name == 'Channels':
                    c.execute("SELECT count(*) FROM Channels WHERE Designation = ?", (values['Designation'],))

                data = c.fetchone()[0]
                if data == 0:
                    values['Source'] = 'Custom'
                    if tab_name == 'Columns':
                        c.execute('''INSERT INTO Columns (Designation,Mass,Area,D,B,tw,T,
                                                        FlangeSlope,R1,R2,Iz,Iy,rz,ry,Zz,Zy,Zpz,Zpy,It,Iw,Source,Type)
                        VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)''',
                                  (values['Designation'], values['Mass'], values['Area'], values['D'],
                                   values['B'], values['tw'], values['T'], values['FlangeSlope'],
                                   values['R1'], values['R2'], values['Iz'], values['Iy'], values['rz'],
                                   values['ry'], values['Zz'], values['Zy'], values['Zpz'], values['Zpy'],
                                   values['It'], values['Iw'], values['Source'], values['Type']))
                    elif tab_name == 'Beams':
                        c.execute('''INSERT INTO Beams (Designation,Mass,Area,D,B,tw,T,
                                                        FlangeSlope,R1,R2,Iz,Iy,rz,ry,Zz,Zy,Zpz,Zpy,It,Iw,Source,Type)
                                                        VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)''',
                                  (values['Designation'], values['Mass'], values['Area'], values['D'],
                                   values['B'], values['tw'], values['T'], values['FlangeSlope'],
                                   values['R1'], values['R2'], values['Iz'], values['Iy'], values['rz'],
                                   values['ry'], values['Zz'], values['Zy'], values['Zpz'], values['Zpy'],
                                   values['It'], values['Iw'], values['Source'], values['Type']))
```

```
                            elif tab_name == 'Angles':
                                c.execute('''INSERT INTO Angles (Designation,Mass,Area,a,b,t,R1,R2,
Cz,Cy,Iz,Iy,Iumax,Ivmin,rz,ry,rumax,rvmin,Zz,Zy,Zpz,Zpy,It,Source,Type)
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)''',
                                          (values['Designation'], values['Mass'], values['Area'], values['a'],
                                           values['b'], values['t'], values['R1'], values['R2'], values['Cz'],
                                           values['Cy'], values['Iz'], values['Iy'], values['Iumax'], values['Ivmin'],
                                           values['rz'], values['ry'], values['rumax'], values['rvmin'], values['Zz'],
                                           values['Zy'], values['Zpz'], values['Zpy'], values['It'], values['Source'],
                                           values['Type']))
                            elif tab_name == 'Channels':
                                c.execute('''INSERT INTO Channels (Designation,Mass,Area,D,B,tw,T,
FlangeSlope,R1,R2,Cy,Iz,Iy,rz,ry,Zz,Zy,Zpz,Zpy,Source,Type)
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)''',
                                          (values['Designation'], values['Mass'], values['Area'], values['D'],
                                           values['B'], values['tw'], values['T'], values['FlangeSlope'], values['R1'],
                                           values['R2'], values['Cy'], values['Iz'], values['Iy'], values['rz'],
                                           values['ry'], values['Zz'], values['Zy'], values['Zpz'], values['Zpy'],
                                           values['Source'], values['Type']))

                            conn.commit()
                            c.close()

                        else:
                            ignored.append(values['Designation'])

                    conn.close()
                    message = QMessageBox()
                    message.setWindowTitle('Successful')
                    message.addButton(message.Ok)
```

```python
                    message.setText('File data is imported successfully to the database.')
                    if discarded or ignored:
                        rejected = message.addButton('Rejected Sections', message.ActionRole)
                        rejected.clicked.connect(lambda: self.import_validation_dialog(discarded, ignored))
                    message.exec()
                else:
                    QMessageBox.information(QMessageBox(), 'Information',
                                            str(str(tab_name) + ' Sheet has headers different than database.'))

            else:
                QMessageBox.information(QMessageBox(), 'Information', str(' File does not contain '+str(tab_name)+' Sheet.'))

        except IOError:
            QMessageBox.information(QMessageBox(), "Unable to open file",
                                    "There was an error opening \"%s\"" % fileName)
            return

```

Appendix K

Code for Import validation

```
    def import_db_validation(self, tab, key, value):

        if key in ['Mass', 'Area', 'D', 'B', 'tw', 'T', 'FlangeSlope', 'R1',
        ↪  'R2',
                   'Iz', 'Iy', 'rz', 'ry', 'Zz', 'Zy','Zpz', 'Zpy', 'It',
                   ↪  'Iw']:
            return isinstance(value, int) or isinstance(value, float)
        else:
            return True

    def import_validation_dialog(self, discarded, ignored):

        dialog = QDialog()
        dialog.setWindowTitle('Rejected Sections')
        vlayout = QVBoxLayout(dialog)
        height = 200
        total = len(discarded)+len(ignored)
        if 0 < total < 30:
            height += total*10
        else:
            height = 500
        dialog.resize(400, height)
        dialog.setLayout(vlayout)
        if discarded:
            scroll_discarded = QScrollArea(dialog)
            vlayout.addWidget(scroll_discarded)
            scroll_discarded.setWidgetResizable(True)
            scroll_discarded.setVerticalScrollBarPolicy(
                QtCore.Qt.ScrollBarAsNeeded)
            widget_discarded = QWidget(scroll_discarded)
            layout_discarded = QVBoxLayout(widget_discarded)
            widget_discarded.setLayout(layout_discarded)
            label_discarded = QLabel("These values were rejected because of
            ↪  validation.")
            layout_discarded.addWidget(label_discarded)
            scroll_discarded.setWidget(widget_discarded)
            text_discarded = QTextBrowser()
            layout_discarded.addWidget(text_discarded)
            for d in discarded:
                text_discarded.append(d)
        if ignored:
            scroll_ignored = QScrollArea(dialog)
            vlayout.addWidget(scroll_ignored)
            scroll_ignored.setWidgetResizable(True)
            scroll_ignored.setVerticalScrollBarPolicy(
                QtCore.Qt.ScrollBarAsNeeded)
            widget_ignored = QWidget(scroll_ignored)
            layout_ignored = QVBoxLayout(widget_ignored)
            widget_ignored.setLayout(layout_ignored)
            label_ignored = QLabel("These values were ignored because they
            ↪  already exist in database.")
            layout_ignored.addWidget(label_ignored)
            scroll_ignored.setWidget(widget_ignored)
            text_ignored = QTextBrowser()
            layout_ignored.addWidget(text_ignored)
            for i in ignored:
                text_ignored.append(i)
        dialog.exec()
```

Appendix L

Code for New features

```
    def change_source(self):

        designation = self[0]
        source = 'Custom'
        if designation in connectdb("Columns", call_type="dropdown"):
            source = get_source("Columns", designation)
        elif designation in connectdb("Beams", call_type="dropdown"):
            source = get_source("Beams", designation)
        elif designation in connectdb("Angles", call_type="dropdown"):
            source = get_source("Angles", designation)
        elif designation in connectdb("Channels", call_type="dropdown"):
            source = get_source("Channels", designation)

        d = {'Label_23': str(source),
             'Label_24': str(source),
             'Label_21': str(source)}
        return d
```

```
def get_source(table_name, designation):

    conn = sqlite3.connect(PATH_TO_DATABASE)

    if table_name == "Angles":
        cursor = conn.execute("SELECT Source FROM Angles WHERE Designation =
        ↪  ?", (designation,))

    elif table_name == "Channels":
        cursor = conn.execute("SELECT Source FROM Channels WHERE Designation =
        ↪  ?", (designation,))

    elif table_name == "Beams":
        cursor = conn.execute("SELECT Source FROM Beams WHERE Designation = ?",
        ↪  (designation,))

    else:
        cursor = conn.execute("SELECT Source FROM Columns WHERE Designation =
        ↪  ?", (designation,))

    source = cursor.fetchone()[0]
    return str(source)
    if lable in [KEY_DISP_FU, KEY_DISP_FY, KEY_DISP_POISSON_RATIO, KEY_DISP_THERMAL_EXP,
        KEY_DISP_MOD_OF_ELAST, KEY_DISP_MOD_OF_RIGID, 'Source']:
        line.setReadOnly(True)
        self.do_not_clear_list.append(line)
    if main.module_name(main) in [KEY_DISP_TENSION_BOLTED, KEY_DISP_TENSION_WELDED] and lable in \
        [KEY_DISP_LOCATION, KEY_DISP_SEC_PROFILE]:
        line.setReadOnly(True)
        self.do_not_clear_list.append(line)
    if last_title == KEY_DISP_DIMENSIONS:
        if element[1] in [KEY_DISP_ROOT_R, KEY_DISP_TOE_R]:
            regex_validator = QtCore.QRegExp("[0-9]*[.][0-9]*|[.][0-9]*|0")
        else:
            regex_validator = QtCore.QRegExp("[1-9][0-9]*[.][0-9]*|[.][0-9]*")
        line.setValidator(QtGui.QRegExpValidator(regex_validator, line))
    if last_title == KEY_DISP_SEC_PROP:
        regex_validator = QtCore.QRegExp("[1-9][0-9]*[.][0-9]*|[.][0-9]*|N/A|-")
        line.setValidator(QtGui.QRegExpValidator(regex_validator, line))

```