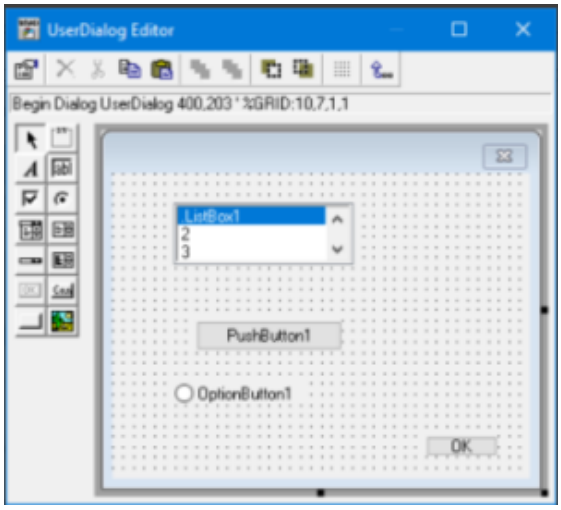
Python GUI with AWRDE

Introduction

An extremely useful capability of the SAX Basic implementation of Visual Basic for Applications (VBA) built into the AWR Design Environment (AWRDE) is creating a graphical user interface (GUI). The UserDialog editor within the AWR Scripting Editor allows for simple drag-and-drop of various widgets such as push buttons, list boxes, radio buttons, etc.



But, what about creating a GUI in Python? The good news is that there are several Python GUI modules available. This article will highlight two of them: **Tkinter** and **PyQt** (Qt for Python). **Tkinter** offers a very basic user interface whereas **PyQt** has a more complete set of GUI widgets available.

The first time through the process, writing GUI code is a fairly daunting task. This is true for any of the methods described in this article. To be fair, the SAX Basic implementation, while simple to construct the dialog boxes using drag-and-drop still has its complications related to event handling. Some templates will be given that can serve as a starting place for creating your first GUI in Python.

This article is intended as a very brief introduction to both **Tkinter** and **PyQt** as it relates to interfacing with AWRDE. For detailed information on the Python GUI module installation and API, please refer to https://www.python.org/doc for **Tkinter** and to this link https://doc.qt.io for **PyQt.**

A prerequisite is that Python, a Python IDE and the **pyawr** module have been installed. See AWR Scripting in Python: Getting Started and Installation for installation instructions. Additionally, Python modules for the GUI will need to be installed and is described below.

Setup

At the beginning of the Python script, the appropriate modules must be installed and a link between Python and AWRDE be established. Place the following code in the script:

pyawr module

```
import pyawr.mwoffice as mwo   #Import pyawr interface to AWRDE
awrde = mwo.CMWOffice()        #Establish link to AWRDE
project = awrde.Project        #Create "project" variable as a convenience item
```

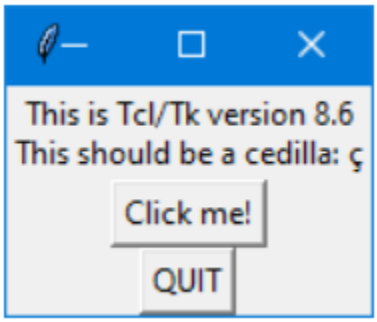
More information on interfacing between Python and AWRDE can be found in AWR Scripting in Python: Using the AWRDE API Scripting Guide

Tkinter

Tkinter is an open-source module that is part of the standard Python install from https://www.python.org To determine if Tkinter is installed in your system, in a command widow type:

```
python -m tkinter
```

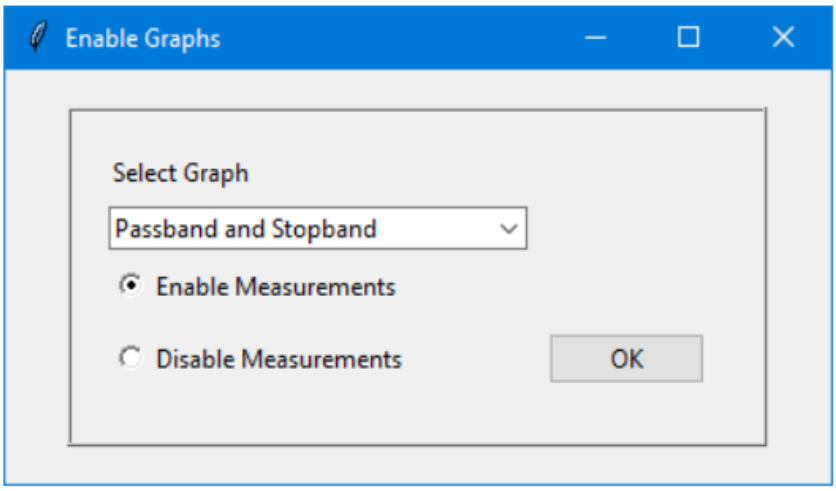
If successful this dialog should appear:



Tkinter offers a modest set of widgets including push buttons, radio buttons, text boxes, list boxes, etc.; a very similar set as what is available in SAX Basic. Due to its wide use, there is a rich set of example code available on the web. Being open-source, there are no licensing restrictions if deployment of your code is a requirement. One drawback with Tkinter as opposed to SAX Basic is lack of dialog box creation using a graphical drag-and-drop method. Tkinter's widgets are all entered by code only.

Example Tkinter Code

A simple example will be used to demonstrate Tkinter coding. The installed example project: LPF_Lumped.emp will be used. This example will create a dialog box that allows the user to select a graph in the project and with that selection, either enable or disable all the measurements within that graph. The dialog box to be constructed will look like:



Import tkinter and its submodule ttk

```
import tkinter as tk
from tkinter import ttk
```

While not strictly necessary, placing all the GUI code within a class may be beneficial. This is especially true if more than one dialog box is needed in the project. A separate class for each dialog box would be used. In this way, the potential for mixing up variables and methods between the different dialog boxes is minimized. So, this first section sets up the class GraphEnable():

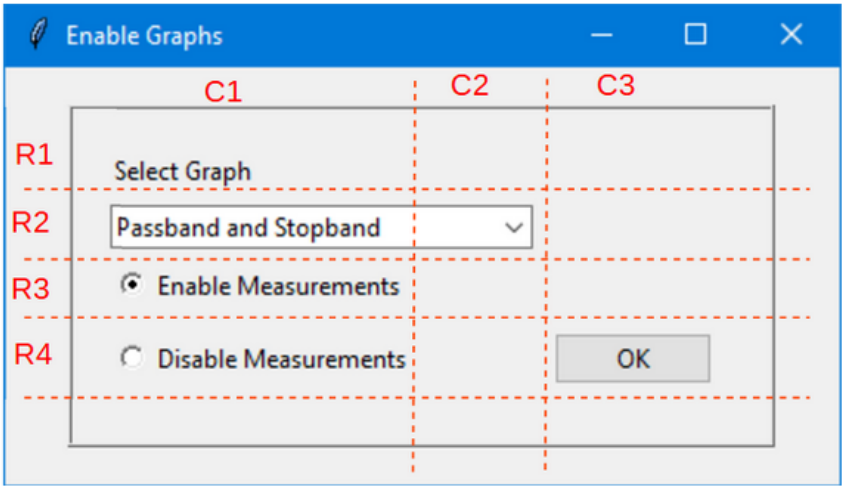
```
class GraphEnable(object):
    def __init__(self):
        pass
    def GraphEnable_ui(self, GraphName_list):
        self.root = tk.Tk()                    #Setup main (root) application window
        self.root.title('Enable Graphs')       #Add title to the window
        self.root.geometry('400x200+400+700')  #width * height + x-position + y-position

                #Allow window to expand if window is resized
        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)

                #Add frame. Padding is margin around the frame (left, top, right, bottom)
        mainframe = ttk.Frame(self.root, padding='10 10 10 10', borderwidth=10, relief='ridge')
        mainframe.grid(column=0, row=0) #Set grid for placing widgets
```

These lines start the method GraphEnable_ui(). The variable **root** will contain methods and variables from the Tk() method. This is the *root* or base window. Next, ensure that the window is expandable if the user tries to re-size the window. Adding a window title, setting the window geometry and adding a frame are next.

Adding the widgets: Combo box, Radio buttons and OK button which exits the dialog box.

```
        #Add Widgets

        #Add label
        ttk.Label(mainframe, text='Select Graph').grid(column=0, row=0, pady=2, sticky=(tk.W))

        #Add Combo Box
        self.GraphComboBox = ttk.Combobox(mainframe, values=GraphName_list, width=30)
        self.GraphComboBox.grid(column=0, row=2, pady=5, sticky=tk.W)
        self.GraphComboBox.current(0)
        self.GraphComboBox.bind('<<ComboboxSelected>>', self.ComboBoxHandler)
        self.GraphSelected = GraphName_list[0]

        #Radiobutton
        self.var = tk.IntVar()
        self.EnableButton = tk.Radiobutton(mainframe, text='Enable Measurements',\
                variable=self.var, value=0, command=self.RadioButtonHandler)
        self.EnableButton.grid(column=0, row=3, sticky=tk.W)
        self.DisableButton = tk.Radiobutton(mainframe, text='Disable Measurements',\
                variable=self.var, value=1, command=self.RadioButtonHandler)
        self.DisableButton.grid(column=0, row=4,  sticky=tk.W)
        self.RadioButtonSelected = 0

        #OK Button. Event driven: call method ok_button() on click
        ttk.Button(mainframe, text='OK', command=self.ok_button).grid(column=2, row=4, padx=10, pady=10)
```

The widgets are positioned in a grid of rows and columns. For this example there are 3 columns and 4 rows as shown:



The .grid() method for each widget has entries for the rows and columns.The sticky= parameter anchors the widget to either north, south, east or west within the designated grid box for that widget. padx= and pady= adds spacing between widgets in either the x or y direction. Events are handled by calls to other methods in the class.

After the widgets are added, then launch the widow.

```
        #Start the window
        self.root.mainloop()   #Start the window. Create infinite do-loop
        self.root.quit()       #Close the window
        return self.GraphSelected, self.RadioButtonSelected
        #
```

The program will remain in an infinite do-loop at the root.mainloop() line. A root.destroy() command will exit the mainloop() which allows the code to continue to the root.quit() command that will close the window.

Here are the other methods within the class GraphEnable(). These are event handlers for the combo box, radio buttons and the OK button.

```
        def ComboBoxHandler(self, event=None):
        self.GraphSelected = self.GraphComboBox.get()
        #
    def RadioButtonHandler(self):
        self.RadioButtonSelected = self.var.get()
        #
    def ok_button(self):#-----------------------------------------
        self.root.destroy()    #Exit the infinite do-loop: root.mainloop()
        #
```

Here is the rest of the code outside of the class.

```
#Setup---------------------------------------------------------------
graph_ui = GraphEnable()

#Main----------------------------------------------------------------
#Get graphs names from project
GraphName_list = list()
for graph in project.Graphs:
    GraphName_list.append(graph.Name)

GraphSelected, RadioButtonSelected = graph_ui.GraphEnable_ui(GraphName_list)
graph = project.Graphs(GraphSelected)
for meas in graph.Measurements:
    if RadioButtonSelected == 0: #Eanbled
        meas.Enabled = True
    elif RadioButtonSelected == 1:
        meas.Enabled = False

#Exit----------------------------------------------------------
print('')
print('Done . . . ')
```

GraphName_list is populated with the names of all the graphs in the AWRDE project. After the GUI exits, the graph variable is assigned to the selected graph name. Then, depending on which radio button was selected, the measurements in the selected graph are either enabled or disabled.

The entire code set is contained here:

GraphEnable Full Code

```
import tkinter as tk
from tkinter import ttk
import pyawr.mwoffice as mwo #Import pyawr interface to AWRDE

class GraphEnable(object):
    def __init__(self):
        pass

    def GraphEnable_ui(self, GraphName_list):
        self.root = tk.Tk()      #Setup main (root) application window
        self.root.title('Enable Graphs')
        self.root.geometry('400x200+400+700')  #width * height + x-position + y-position

        #Allow window to expand if window is resized
        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)

        #Add frame. Padding is margin around the frame (left, top, right, bottom)
        mainframe = ttk.Frame(self.root, padding='10 10 10 10', borderwidth=10, relief='ridge')
        mainframe.grid(column=0, row=0) #Set grid for placing widgets

        #Add Widgets

        #Add label
        ttk.Label(mainframe, text='Select Graph').grid(column=0, row=0, pady=2, sticky=(tk.W))

        #Add Combo Box
```

```
        self.GraphComboBox = ttk.Combobox(mainframe, values=GraphName_list, width=30)
        self.GraphComboBox.grid(column=0, row=2, pady=5, sticky=tk.W)
        self.GraphComboBox.current(0)
        self.GraphComboBox.bind('<<ComboboxSelected>>', self.ComboBoxHandler)
        self.GraphSelected = GraphName_list[0]

        #Radiobutton
        self.var = tk.IntVar()
        self.EnableButton = tk.Radiobutton(mainframe, text='Enable Measurements',\
                variable=self.var, value=0, command=self.RadioButtonHandler)
        self.EnableButton.grid(column=0, row=3, sticky=tk.W)
        self.DisableButton = tk.Radiobutton(mainframe, text='Disable Measurements',\
                variable=self.var, value=1, command=self.RadioButtonHandler)
        self.DisableButton.grid(column=0, row=4,  sticky=tk.W)
        self.RadioButtonSelected = 0


        #OK Button. Event driven: call method ok_button() on click
        ttk.Button(mainframe, text='OK', command=self.ok_button).grid(column=2, row=4, padx=10, pady=10)

        #Start the window
        self.root.mainloop()   #Start the window. Create infinite do-loop
        self.root.quit()       #Close the window
        return self.GraphSelected, self.RadioButtonSelected
        #
    def ComboBoxHandler(self, event=None):
        self.GraphSelected = self.GraphComboBox.get()
        #
    def RadioButtonHandler(self):
        self.RadioButtonSelected = self.var.get()
        #
    def ok_button(self):#-----------------------------------------
        self.root.destroy()    #Exit the infinite do-loop: root.mainloop()
        #

#Setup---------------------------------------------------------------
graph_ui = GraphEnable()
version_str = '16.0' #Set specific version of AWRDE
awrde = mwo.CMWOffice(version=version_str)  #Establish link to AWRDE
project = awrde.Project #Create "project" variable as a convenience item
#
#Main----------------------------------------------------------------
#Get graphs names from project
GraphName_list = list()
for graph in project.Graphs:
    GraphName_list.append(graph.Name)

GraphSelected, RadioButtonSelected = graph_ui.GraphEnable_ui(GraphName_list)
graph = project.Graphs(GraphSelected)
for meas in graph.Measurements:
    if RadioButtonSelected == 0: #Eanbled
        meas.Enabled = True
    elif RadioButtonSelected == 1:
        meas.Enabled = False

#Exit----------------------------------------------------------
print('')
print('Done . . . ')
```

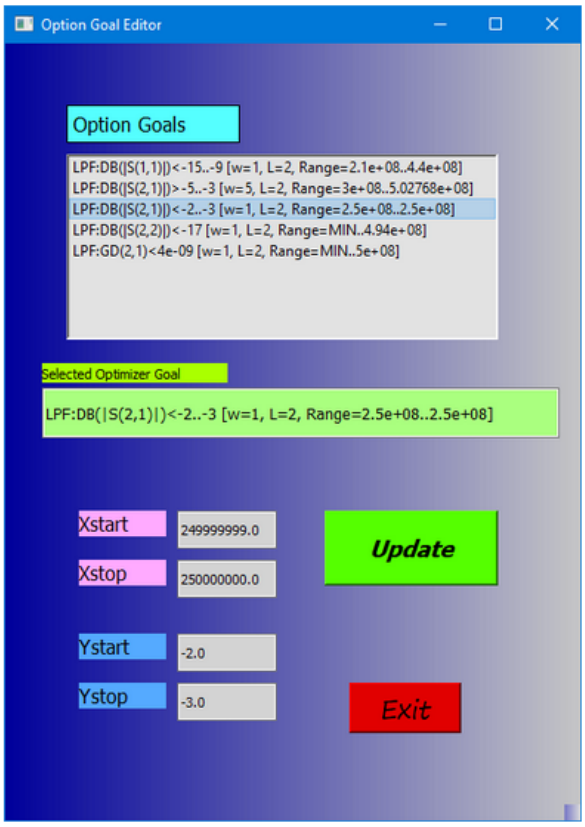
PyQt

PyQt enables more functionality and an enhanced look and feel over what Tkinter can provide. Optionally, and highly recommended, the **Qt Designer** tool can be used to graphically design your dialog box much in the same way as the UserDialog editor in SAX Basic. **Qt Designer** creates an XML file that is then converted into Python code using a Windows cmd instruction. This Python code, stored as a Python module, can then be imported into your Python script the same as any standard module. See Qt Designer link for installation and this link on how to operate **Qt Designer**.

Certain licensing restrictions apply for the use of PyQt. Please consult PyQt Licensing before considering PyQt for GUI development.

Example PyQt Code

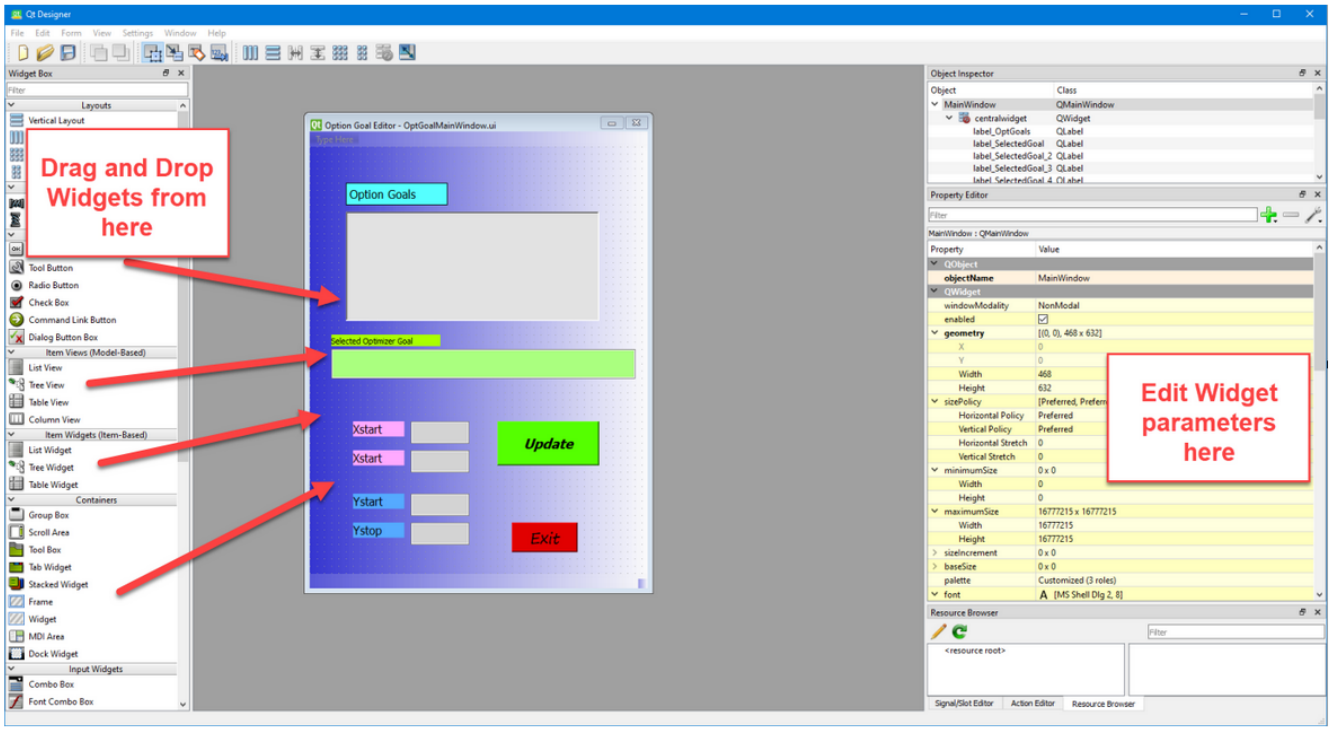
This example will read Optimizer Goals from an AWRDE project and allow editing of the x start/stop as well as the y start/stop values:



The installed example project LPF_Lumped.emp is a good project that can be used.

For this GUI design, **Qt Designer** will be used to graphically construct the user interface. However, **Qt Designer** is not a requirement as the user interface could be constructed using code only. Constructing by code or using the graphical approach is a matter of personal preference.

Shown here is the **Qt Designer** panel for this design. The construction is fairly straight forward: Drag-and-drop widgets from the Widget Box panel and customize each widget in the Property Editor panel. Colors, fonts, borders and even assigning an image to a widget (for example a push button) are all editing choices.



Qt Designer saves its files with the extension **.ui** The following steps involve creating a Python (.py) file and it is highly recommended that the **Qt Designer .ui** file be saved in the same directory as the Python file. A command line is used to convert the **.ui** file into a Python module. In either a command window or PowerShell window, cd to the directory where the **.ui** file is stored and type in a command similar to what is shown here:

```
pyuic5 -o .\main_window.py .\OptGoalMainWindow.ui
```

This example uses PyQt5; if PyQt6 is being used then the command would be *pyuic6.* The .py file is one that will be created and the .ui file is the Qt Designer's output file. Note that this process is a one way event. You cannot make edits in the Python GUI code and have them show up in **Qt Designer**. To make edits to the GUI, you need to go back the Qt Designer, make the edits and then use the command line to create a new version of the Python GUI code.

The structure of the Python file created by the above process is as shown:

Main Window Python Module

```
# Form implementation generated from reading ui file '.\OptGoalMainWindow.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(468, 632)
        MainWindow.setStyleSheet("background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:0, stop:0
rgba(0, 0, 200, 100), stop:1 rgba(255, 255, 255,100));")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton_exit = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_exit.setGeometry(QtCore.QRect(280, 520, 91, 41))
        self.pushButton_exit.setStyleSheet("font: 75 16pt \"Segoe Print\";\n"
"background-color: rgb(225, 0, 0);")
        self.pushButton_exit.setObjectName("pushButton_exit")
        self.label_OptGoals = QtWidgets.QLabel(self.centralwidget)
        self.label_OptGoals.setGeometry(QtCore.QRect(50, 50, 141, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
```

This is a Python file that acts as a module. All the Python code needed to create and customize the overall container and all of its widgets is found here. In **Qt Designer**, the overall container was given the name MainWindow, so the first class in this module is given the name Ui_<overall container name>. For this example: Ui_MainWindow. Also notice the method setupUi() in this class. The class and setupUi() method will be referred to in the next section.

The next step is to create the top level Python code.

First import the required modules and establish a link to AWRDE

```
import pyawr.mwoffice as mwo #Import pyawr interface to AWRDE

#Import PyQt modules
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QLineEdit
#
#Import Qt Designer generated module
from main_window import Ui_MainWindow
#
#Establish AWRDE Link
version_str = '16.0'                        #Set specific version of AWRDE
awrde = mwo.CMWOffice(version=version_str)  #Establish link to AWRDE
project = awrde.Project                     #Create "project" variable as a convenience item
#
```

Line 7 shows importing the Ui class from the Qt Designer created Python module.

When writing PyQt code it is almost a necessity to write the code in object oriented fashion using classes. Here the class MainWindow is where the GUI event handlers are contained:

```
class MainWindow(Ui_MainWindow, QMainWindow, QWidget):
    def __init__(self):
        super().__init__()     #Initialize inherited classes
        self.setupUi(self)     #setupUi() method is in the Qt Designer generated class Ui_MainWindow
        self.ConnectSignals()  #Call to event handler method
        self.GetOptGoals()     #Read Optimizer Goals from AWRDE project
        self.WriteListWidget() #Write Optimizer Goals to the list widget
        #
    def ConnectSignals(self):
        #
        #Event handlers
        #
        self.pushButton_exit.clicked.connect(self.ExitApplication)  #Exit Pushbutton
        self.pushButton_update.clicked.connect(self.Update)         #Update Pushbutton
        self.listWidget_OptGoals.itemClicked.connect(self.ListWidetHandler) #listWidget
        #
```

Class inheritance is used to connect the MainWindow class to classes from Qt as well as the Python code generated by Qt Designer (this is the Ui_MainWindow class). On line 17, the call to the setupUi() method in the Qt Designer generated Python module is called. All the links to the GUI widgets emanate from this call. Line 18 calls the method ConnectSignals() which creates all the event handlers (called signals and slots in Qt nomenclature). The next lines a specific to this application.

After that, the code waits for an event (button click) to take place.

Here is the code called when the Exit button is pressed:

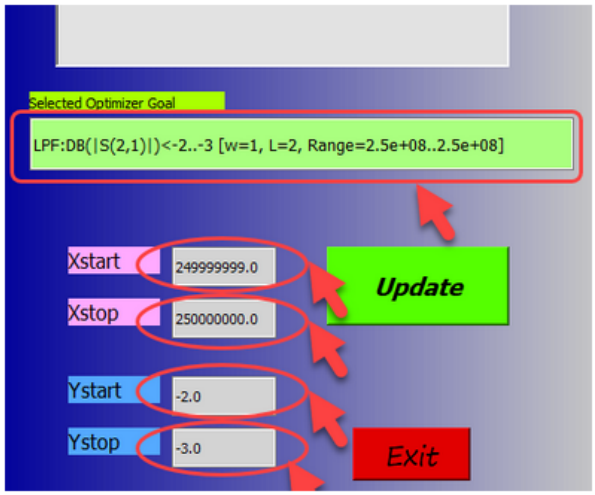
```
 def ExitApplication(self):
        #
        #Called when the Exit push button has been clicked
        #
        self.close()  #Close the main window and exit the class
        #
```

close() function causes the main window to stop and then forces the code to exit the MainWindow class

Here is the code when one of the line items is selected in the line widget:

```
def ListWidetHandler(self, item):
        #
        #Called when one of the line items has been selected
        #
        self.CurrentGoalText = item.text()         #Get text from selected line item
        self.CurrentGoal_idx = self.listWidget_OptGoals.currentRow() #Get selected line index
        #
        self.lineEdit_CurrentGoal.setText(self.CurrentGoalText) #Update line edit widget
        #
        Xstart = self.OptGoal_list[self.CurrentGoal_idx][1]
        Xstop = self.OptGoal_list[self.CurrentGoal_idx][2]
        Ystart = self.OptGoal_list[self.CurrentGoal_idx][3]
        Ystop = self.OptGoal_list[self.CurrentGoal_idx][4]
        #
        #Update X,Y start and stop line edit widgets
        self.lineEdit_Xstart.setText(str(Xstart))
        self.lineEdit_Xstop.setText(str(Xstop))
        self.lineEdit_Ystart.setText(str(Ystart))
        self.lineEdit_Ystop.setText(str(Ystop))
        #
```

lineEdit widgets are highlighted below and are uses for single line text editing and display



Here is the code when the Update button is pressed:

```
def Update(self):
        #
        #Called when the Update push button has been clicked
        #
        #Read the data from X,Y start and stop line edit widgets and write values
        #to AWRDE Optimizer Goal properties
        #
        Xstart = self.lineEdit_Xstart.text()
        project.OptGoals(self.CurrentGoal_idx+1).xStart = float(Xstart)
        Xstop = self.lineEdit_Xstop.text()
        project.OptGoals(self.CurrentGoal_idx+1).xStop = float(Xstop)
        Ystart = self.lineEdit_Ystart.text()
        project.OptGoals(self.CurrentGoal_idx+1).yStart = float(Ystart)
        Ystop = self.lineEdit_Ystop.text()
        project.OptGoals(self.CurrentGoal_idx+1).yStop = float(Ystop)
        self.GetOptGoals()
        self.WriteListWidget()
        self.lineEdit_CurrentGoal.clear()
        #
```

lineEdit text is read and then sent along to AWRDE Optimizer Goal properties

Outside of the MainWidow class, here is the code to start the MainWindow class

```
#Setup:----------------------------------------------------------------
app = QApplication([])         #Create the Qt application
window = MainWindow()          #Create GUI window
window.show()                  #Display the Main Window

#Main------------------------------------------------------------------
app.exec()                    #Create infinite loop until close() command is sent

#Exit------------------------------------------------------------------------
print('Done . . .')
```

Complete code for the above PyQt example is contained here:

PyQt Example Full Code

```
import pyawr.mwoffice as mwo #Import pyawr interface to AWRDE

#Import PyQt modules
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QLineEdit
```

```
#
#Import Qt Designer generated module
from main_window import Ui_MainWindow
#
#Establish AWRDE Link
version_str = '16.0'                        #Set specific version of AWRDE
awrde = mwo.CMWOffice(version=version_str)  #Establish link to AWRDE
project = awrde.Project                     #Create "project" variable as a convenience item
#
class MainWindow(Ui_MainWindow, QMainWindow, QWidget):
    def __init__(self):
        super().__init__()     #Initialize inherited classes
        self.setupUi(self)     #setupUi() method is in the Qt Designer generated class Ui_MainWindow
        self.ConnectSignals()  #Call to event handler method
        self.GetOptGoals()     #Read Optimizer Goals from AWRDE project
        self.WriteListWidget() #Write Optimizer Goals to the list widget
        #
    def ConnectSignals(self):
        #
        #Event handlers
        #
        self.pushButton_exit.clicked.connect(self.ExitApplication)  #Exit Pushbutton
        self.pushButton_update.clicked.connect(self.Update)         #Update Pushbutton
        self.listWidget_OptGoals.itemClicked.connect(self.ListWidetHandler) #listWidget
        #
    def ListWidetHandler(self, item):
        #
        #Called when one of the line items has been selected
        #
        self.CurrentGoalText = item.text()         #Get text from selected line item
        self.CurrentGoal_idx = self.listWidget_OptGoals.currentRow() #Get selected line index
        #
        self.lineEdit_CurrentGoal.setText(self.CurrentGoalText) #Update line edit widget
        #
        Xstart = self.OptGoal_list[self.CurrentGoal_idx][1]
        Xstop = self.OptGoal_list[self.CurrentGoal_idx][2]
        Ystart = self.OptGoal_list[self.CurrentGoal_idx][3]
        Ystop = self.OptGoal_list[self.CurrentGoal_idx][4]
        #
        #Update X,Y start and stop line edit widgets
        self.lineEdit_Xstart.setText(str(Xstart))
        self.lineEdit_Xstop.setText(str(Xstop))
        self.lineEdit_Ystart.setText(str(Ystart))
        self.lineEdit_Ystop.setText(str(Ystop))
        #
    def WriteListWidget(self):
        #
        #Populate the listWidget with all the Optimizer Goals in the AWRDE project
        #
        self.listWidget_OptGoals.clear()
        for i in range(len(self.OptGoal_list)):
            self.listWidget_OptGoals.addItem(self.OptGoal_list[i][0])
        #
    def Update(self):
        #
        #Called when the Update push button has been clicked
        #
        #Read the data from X,Y start and stop line edit widgets and write values
        #to AWRDE Optimizer Goal properties
        #
        Xstart = self.lineEdit_Xstart.text()
        project.OptGoals(self.CurrentGoal_idx+1).xStart = float(Xstart)
        Xstop = self.lineEdit_Xstop.text()
        project.OptGoals(self.CurrentGoal_idx+1).xStop = float(Xstop)
        Ystart = self.lineEdit_Ystart.text()
        project.OptGoals(self.CurrentGoal_idx+1).yStart = float(Ystart)
        Ystop = self.lineEdit_Ystop.text()
        project.OptGoals(self.CurrentGoal_idx+1).yStop = float(Ystop)
        self.GetOptGoals()
        self.WriteListWidget()
        self.lineEdit_CurrentGoal.clear()
```

```
        #
    def GetOptGoals(self):
        #
        #Read the Optimizer Goals from the AWRDE project
        #
        self.OptGoal_list = list()
        for OptGoal in project.OptGoals:
            self.OptGoal_list.append([OptGoal.Name, OptGoal.xStart, OptGoal.xStop,\
                                      OptGoal.yStart, OptGoal.yStop])
    def ExitApplication(self):
        #
        #Called when the Exit push button has been clicked
        #
        self.close()   #Close the main window and exit the class
        #
#Setup:---------------------------------------------------------------
app = QApplication([])         #Create the Qt application
window = MainWindow()          #Create GUI window
window.show()                  #Display the Main Window

#Main----------------------------------------------------------------
app.exec()                    #Create infinite loop until close() command is sent

#Exit----------------------------------------------------------------------
print('Done . . .')
```

GUI code generated by **Qt Designer:**

main_window.py Full Code

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file '.\OptGoalMainWindow.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(468, 632)
        MainWindow.setStyleSheet("background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:0, stop:0
rgba(0, 0, 200, 100), stop:1 rgba(255, 255, 255,100));")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton_exit = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_exit.setGeometry(QtCore.QRect(280, 520, 91, 41))
        self.pushButton_exit.setStyleSheet("font: 75 16pt \"Segoe Print\";\n"
"background-color: rgb(225, 0, 0);")
        self.pushButton_exit.setObjectName("pushButton_exit")
        self.label_OptGoals = QtWidgets.QLabel(self.centralwidget)
        self.label_OptGoals.setGeometry(QtCore.QRect(50, 50, 141, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_OptGoals.setFont(font)
        self.label_OptGoals.setStyleSheet("background-color: rgb(85, 255, 255);")
        self.label_OptGoals.setFrameShape(QtWidgets.QFrame.Box)
        self.label_OptGoals.setFrameShadow(QtWidgets.QFrame.Plain)
        self.label_OptGoals.setLineWidth(1)
        self.label_OptGoals.setObjectName("label_OptGoals")
        self.lineEdit_CurrentGoal = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_CurrentGoal.setGeometry(QtCore.QRect(30, 280, 421, 41))
        font = QtGui.QFont()
```

```
        font.setPointSize(10)
        self.lineEdit_CurrentGoal.setFont(font)
        self.lineEdit_CurrentGoal.setStyleSheet("background-color: rgb(170, 255, 127);")
        self.lineEdit_CurrentGoal.setObjectName("lineEdit_CurrentGoal")
        self.label_SelectedGoal = QtWidgets.QLabel(self.centralwidget)
        self.label_SelectedGoal.setGeometry(QtCore.QRect(30, 260, 151, 16))
        self.label_SelectedGoal.setStyleSheet("background-color: rgb(170, 255, 0);")
        self.label_SelectedGoal.setObjectName("label_SelectedGoal")
        self.listWidget_OptGoals = QtWidgets.QListWidget(self.centralwidget)
        self.listWidget_OptGoals.setGeometry(QtCore.QRect(50, 90, 351, 151))
        self.listWidget_OptGoals.setStyleSheet("background-color: rgb(226, 226, 226);\n"
"border-color: rgb(255, 0, 0);")
        self.listWidget_OptGoals.setFrameShape(QtWidgets.QFrame.WinPanel)
        self.listWidget_OptGoals.setLineWidth(5)
        self.listWidget_OptGoals.setMidLineWidth(4)
        self.listWidget_OptGoals.setEditTriggers(QtWidgets.QAbstractItemView.DoubleClicked|QtWidgets.
QAbstractItemView.EditKeyPressed|QtWidgets.QAbstractItemView.SelectedClicked)
        self.listWidget_OptGoals.setObjectName("listWidget_OptGoals")
        self.lineEdit_Xstart = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_Xstart.setGeometry(QtCore.QRect(140, 380, 81, 31))
        self.lineEdit_Xstart.setStyleSheet("background-color: rgb(211, 211, 211);")
        self.lineEdit_Xstart.setObjectName("lineEdit_Xstart")
        self.label_SelectedGoal_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_SelectedGoal_2.setGeometry(QtCore.QRect(60, 380, 71, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_SelectedGoal_2.setFont(font)
        self.label_SelectedGoal_2.setStyleSheet("background-color: rgb(255, 170, 255);")
        self.label_SelectedGoal_2.setObjectName("label_SelectedGoal_2")
        self.lineEdit_Xstop = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_Xstop.setGeometry(QtCore.QRect(140, 420, 81, 31))
        self.lineEdit_Xstop.setStyleSheet("background-color: rgb(211, 211, 211);")
        self.lineEdit_Xstop.setObjectName("lineEdit_Xstop")
        self.label_SelectedGoal_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_SelectedGoal_3.setGeometry(QtCore.QRect(60, 420, 71, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_SelectedGoal_3.setFont(font)
        self.label_SelectedGoal_3.setStyleSheet("background-color: rgb(255, 170, 255);")
        self.label_SelectedGoal_3.setObjectName("label_SelectedGoal_3")
        self.lineEdit_Ystart = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_Ystart.setGeometry(QtCore.QRect(140, 480, 81, 31))
        self.lineEdit_Ystart.setStyleSheet("background-color: rgb(211, 211, 211);")
        self.lineEdit_Ystart.setObjectName("lineEdit_Ystart")
        self.label_SelectedGoal_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_SelectedGoal_4.setGeometry(QtCore.QRect(60, 480, 71, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_SelectedGoal_4.setFont(font)
        self.label_SelectedGoal_4.setStyleSheet("background-color: rgb(85, 170, 255);")
        self.label_SelectedGoal_4.setObjectName("label_SelectedGoal_4")
        self.lineEdit_Ystop = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_Ystop.setGeometry(QtCore.QRect(140, 520, 81, 31))
        self.lineEdit_Ystop.setStyleSheet("background-color: rgb(211, 211, 211);")
        self.lineEdit_Ystop.setObjectName("lineEdit_Ystop")
        self.label_SelectedGoal_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_SelectedGoal_5.setGeometry(QtCore.QRect(60, 520, 71, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_SelectedGoal_5.setFont(font)
        self.label_SelectedGoal_5.setStyleSheet("background-color: rgb(85, 170, 255);")
        self.label_SelectedGoal_5.setObjectName("label_SelectedGoal_5")
        self.pushButton_update = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_update.setGeometry(QtCore.QRect(260, 380, 141, 61))
        font = QtGui.QFont()
        font.setPointSize(14)
        font.setBold(True)
        font.setItalic(True)
        font.setWeight(75)
        self.pushButton_update.setFont(font)
        self.pushButton_update.setStyleSheet("background-color: rgb(85, 255, 0);")
```

```
        self.pushButton_update.setObjectName("pushButton_update")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 468, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Option Goal Editor"))
        self.pushButton_exit.setText(_translate("MainWindow", "Exit"))
        self.label_OptGoals.setText(_translate("MainWindow", "Option Goals"))
        self.label_SelectedGoal.setText(_translate("MainWindow", "Selected Optimizer Goal"))
        self.label_SelectedGoal_2.setText(_translate("MainWindow", "Xstart"))
        self.label_SelectedGoal_3.setText(_translate("MainWindow", "Xstop"))
        self.label_SelectedGoal_4.setText(_translate("MainWindow", "Ystart"))
        self.label_SelectedGoal_5.setText(_translate("MainWindow", "Ystop"))
        self.pushButton_update.setText(_translate("MainWindow", "Update"))
```