

Requirement 16 Report - Decide on a UI Library

Originally, the team planned on using Overwolf to create a UI to interact with our AI agent. In sprint 2, we discovered this was not possible and began exploring alternative UI libraries. To move forward with UI development, the team needs to settle on a specific UI library.

Selected Library: **PyQt**

- Better visuals.
- Better code.
- Standalone GUI Designer
- Extensive Documentation and Tutorials

In the initial research, we limited our options to five libraries, PySimpleGUI, Toga, tKinter, PyQt, and WxPython. Our optimal choice for a GUI revolves around offering basic functionalities, speed and ease of implementation, and appealing design / appearance.

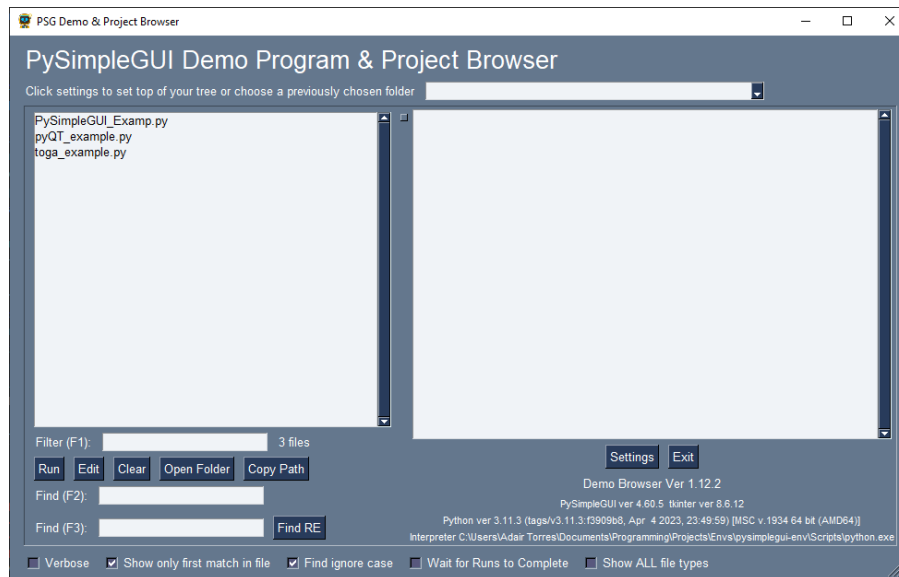
Based upon our initial research and the consensus of Python developer communities, the libraries that stand out the most are PySimpleGUI, Toga, and PyQt. PySimpleGUI is another form of tKinter and WxPython, but the only difference worth noting is that standalone tKinter tends to rank higher than PySimpleGUI. For each of these GUI frameworks, a short explanation and sample GUI will be provided. The sample GUI's are obtained from each framework's webpage or GitHub section dedicated to demonstrations.

For the sake of visual appeal, each review / section will begin on a new page.

PySimpleGUI

PySimpleGui implements "ports" of tKinter, PyQt, and WxPython within itself. While most ports besides tKinter are not fully implemented, they still offer an extensive amount of functionality. As far as development resources, PySimpleGui offers a user's manual, a "cookbook" of different GUI examples, 170+ demo programs, and docstrings that grant help directly from Python or an IDE.

PySimpleGUI's appearance can be appealing, however many of its widgets don't have variety in their shape, and can make the GUI feel very "outdated." Furthermore, the themes offered by PySimpleGUI don't change much aside from the color palette, leaving much to be desired in making a GUI feel unique or different from other apps.



Coding a GUI using PySimpleGUI also does not look very appealing. Rather than use methods for abstraction, PySimpleGUI seems to force the developer into using lists of objects to define sections and controls. The code for the example above is over 700 lines of code, however it is worth noting that its functionality makes up for most of it and is a fully implemented file browser.

```
left_col = sg.Column([
    [sg.ListBox(values=get_file_list(), select_mode=sg.SELECT_MODE_EXTENDED, size=(50,20), bind_return_key=True, key='-DEMO LIST-', expand_x=True, expand_y=True)],
    [sg.Text('Filter (F1):', tooltip=filter_tooltip), sg.Input(size=(25, 1), focus=True, enable_events=True, key='-FILTER-', tooltip=filter_tooltip),
    sg.T(size=(15,1), k='-FILTER NUMBER-')],
    [sg.Button('Run'), sg.B('Edit'), sg.B('Clear'), sg.B('Open Folder'), sg.B('Copy Path')],
    [sg.Text('Find (F2):', tooltip=find_tooltip), sg.Input(size=(25, 1), enable_events=True, key='-FIND-', tooltip=find_tooltip),
    sg.T(size=(15,1), k='-FIND NUMBER-')],
], element_justification='l', expand_x=True, expand_y=True)

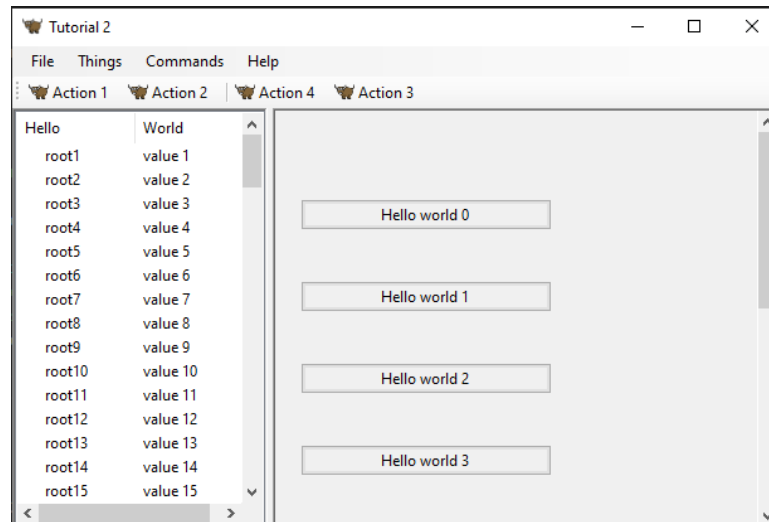
lef_col_find_re = sg.pin(sg.Col([
    [sg.Text('Find (F3):', tooltip=find_re_tooltip), sg.Input(size=(25, 1), key='-FIND RE-', tooltip=find_re_tooltip), sg.B('Find RE')], k='-RE COL-')])

right_col = [
    [sg.Multiline(size=(70, 21), write_only=True, expand_x=True, expand_y=True, key=ML_KEY, reroute_stdout=True, echo_stdout_stderr=True, reroute_cprint=True)],
    [sg.B('Settings'), sg.Button('Exit')],
    [sg.T('Demo Browser Ver ' + __version__)],
    [sg.T('PySimpleGUI ver ' + sg.version.split(' ')[0] + ' tkinter ver ' + sg.tclversion_detailed, font='Default 8', pad=(0,0))],
    [sg.T('Python ver ' + sys.version, font='Default 8', pad=(0,0))],
    [sg.T('Interpreter ' + sg.execute_py_get_interpreter(), font='Default 8', pad=(0,0))],
]

options_at_bottom = sg.pin(sg.Column([sg.CB('Verbose', enable_events=True, k='-VERBOSE-', tooltip='Enable to see the matches in the right hand column'),
sg.CB('Show only first match in file', default=True, enable_events=True, k='-FIRST MATCH ONLY-', tooltip='Disable to see ALL matches found in files'),
sg.CB('Find ignore case', default=True, enable_events=True, k='-IGNORE CASE-'),
sg.CB('Wait for Runs to Complete', default=False, enable_events=True, k='-WAIT-'),
sg.CB('Show ALL file types', default=not python_only, enable_events=True, k='-SHOW ALL FILES-'),
]),
pad=(0,0), k='-OPTIONS BOTTOM-', expand_x=True, expand_y=False, expand_x=True, expand_y=False)
```

Toga

Toga is Python native and OS native, meaning its theme changes to match the OS it is running on. While this isn't a bad choice, it can offer inconsistencies in the design and how the developer wants their GUI to appear across different platforms. Toga's docs webpage doesn't offer too much in the form of tutorials, mainly having four in building a simple text input, using a SplitContainer, building a browser, and drawing on a canvas.



Toga's appearance isn't too great, but isn't too bad either. With widgets changing their appearance based on the OS, technically all the apps will look OS-native, however that does not mean the GUI will match the look of a "modern" OS, only that it will use compatible widgets designed for that OS. Toga's code also follows the same pattern of having the developer implement designs through lists, but not to the same degree and PySimpleGUI, and is generally more readable.

```
right_container.content = right_content

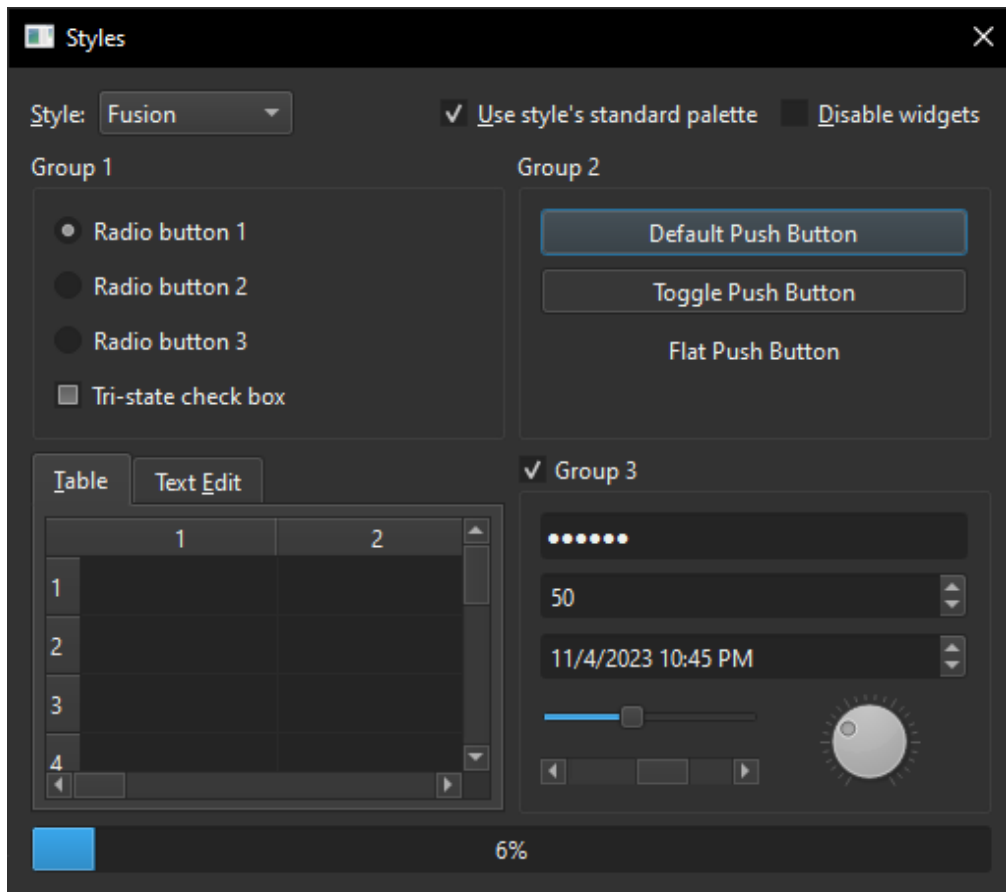
split = toga.SplitContainer()

# The content of the split container can be specified as a simple list:
# split.content = [left_container, right_container]
# but you can also specify "weight" with each content item, which will
# set an initial size of the columns to make a "heavy" column wider than
# a narrower one. In this example, the right container will be twice
# as wide as the left one.
split.content = [(left_container, 1), (right_container, 2)]

# Create a "Things" menu group to contain some of the commands.
# No explicit ordering is provided on the group, so it will appear
# after application-level menus, but *before* the Command group.
# Items in the Things group are not explicitly ordered either, so they
# will default to alphabetical ordering within the group.
things = toga.Group("Things")
cmd0 = toga.Command(
    action0,
    text="Action 0",
    tooltip="Perform action 0",
    icon=brutus_icon,
    group=things,
)
```

PyQt

This will look at PyQt6 in particular, but PyQt5 is also a notable version. Things may start sounding biased here, but that's because PyQt handles the points mentioned thus far for previous GUI frameworks so differently. First off is PyQt's appearance and design. While its colors change with each theme, the appearance and shapes of its widgets also drastically change. The themes still typically look generic, but that can go for almost any framework. However, PyQt offers more appealing and "modern" looking themes alongside themes that look like they were ripped from WindowsXP.

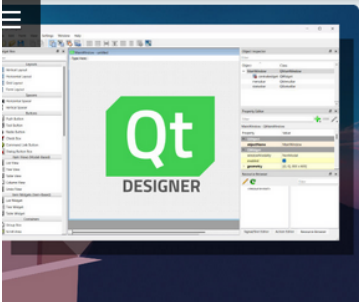


PyQt doesn't just offer "modern" themes. While the code written for it could technically be reduced down to a bunch of lists with widget details, PyQt allows for far more readable code. Or, you could ditch BOTH of these and just use the *standalone Qt Designer*. Looking at PyQt's webpage, this framework feels like it was developed with designers AND developers in mind. The same webpage not only offers extensive documentation on using PyQt and its widgets, but also multiple tutorials on using the standalone to implement basic GUIs, custom widgets, and custom dialogs.

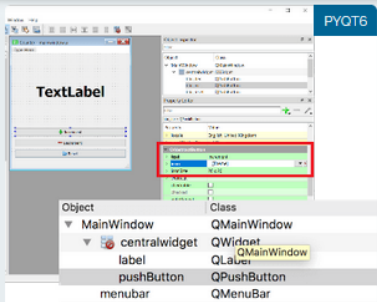
(See example code and tutorials on next page).

```
70 self.createTopLeftGroupBox()
71 self.createTopRightGroupBox()
72 self.createBottomLeftTabWidget()
73 self.createBottomRightGroupBox()
74 self.createProgressBar()
75
76 styleComboBox.textActivated.connect(self.changeStyle)
77 self.useStylePaletteCheckBox.toggled.connect(self.changePalette)
78 disableWidgetsCheckBox.toggled.connect([self.topLeftGroupBox.setDisabled])
79 disableWidgetsCheckBox.toggled.connect(self.topRightGroupBox.setDisabled)
80 disableWidgetsCheckBox.toggled.connect(self.bottomLeftTabWidget.setDisabled)
81 disableWidgetsCheckBox.toggled.connect(self.bottomRightGroupBox.setDisabled)
82
83 topLayout = QHBoxLayout()
84 topLayout.addWidget(styleLabel)
85 topLayout.addWidget(styleComboBox)
86 topLayout.addStretch(1)
87 topLayout.addWidget(self.useStylePaletteCheckBox)
```

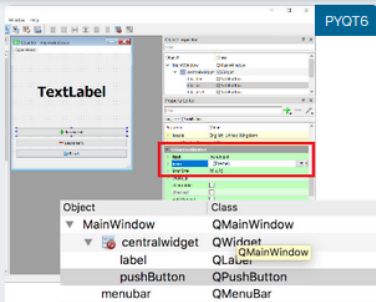
Ln 78, Col 81 Spaces: 4 UTF-8 CRLF Python 3.11.3 (pysimplegui-env)



Install Qt Designer Standalone
Qt Designer Download for Windows, Mac and Linux



PyQt6 Tutorial
Embedding custom widgets from Qt Designer
Learn how to use custom widgets in your PyQt6



PyQt6 Tutorial
Creating Dialogs With Qt Designer
Using the drag and drop editor to build PyQt6 dialogs