Chapter 8
The wxPython GUI Library

8.1 The wxPython Library

The wxPython library is a cross platform GUI library (or toolkit) for Python. It allows programmers to develop highly graphical user interfaces for their programs using common concepts such as menu bars, menus, buttons, fields, panels and frames.

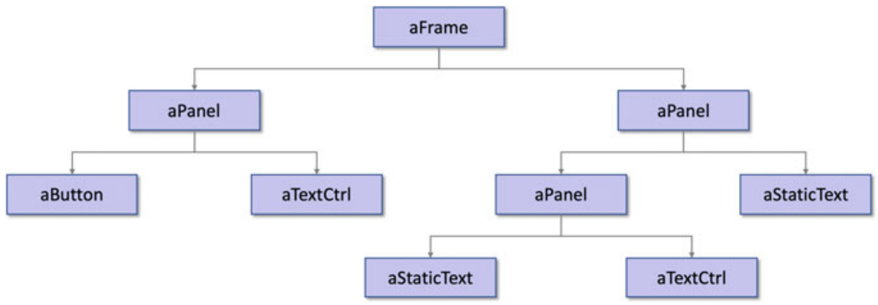
In wxPython all the elements of a GUI are contained within top level windows such as a `wx.Frame` or a `wx.Dialog`. These windows contain graphical components known as widgets or controls. These widgets/controls may be grouped together into Panels (which may or may not have a visible representation).

Thus in wxPython we might construct a GUI from:

- **Frames** *which* provide the basic structure for a window: borders, a label and some basic functionality (e.g. resizing).
- **Dialogs** which are like Frames but provide fewer border controls.
- **Widgets/Controls** that are graphical objects displayed in a frame. Some other languages refer to them as UI components. Examples of widgets are buttons, checkboxes, selection lists, labels and text fields.
- **Containers** are component that are made up of one or more other components (or containers). All the components within a container (such as a panel) can be treated as a single entity.

Thus a GUI is constructed hierarchically from a set of widgets, containers and one or more Frames (or in the case of a pop up dialog then Dialogs). This is illustrated below for a window containing several panels and widgets:

J. Hunt, *Advanced Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-25943-3_8



Windows such as Frames and Dialogs have a component hierarchy that is used (amongst other things) to determine how and when elements of the window are drawn and redrawn. The component hierarchy is rooted with the frame, within which components and containers can be added.

The above figure illustrates a component hierarchy for a frame, with two container Panels and a few basic widgets/ui components held within the Panels. Note that a panel can contain another sub panel with different widgets in.

8.1.1 wxPython Modules

The wxPython library is comprised of many different modules. These modules provide different features from the core `wx` module to the html oriented `wx.html` and `wx.html2` modules. These modules include:

- `wx` which holds the core widgets and classes in the `wx` library.
- `wx.adv` that provides less commonly used or more advanced widgets and classes.
- `wx.grid` contains widgets and classes supporting the display and editing of tabular data.
- `wx.richtext` consists of widgets and classes used for displaying multiple text styles and images.
- `wx.html` comprises widgets and supporting classes for a generic html renderer.
- `wx.html2` provides further widget and supporting classes for a native html renderer, with CSS and javascript support.

8.1.2 Windows as Objects

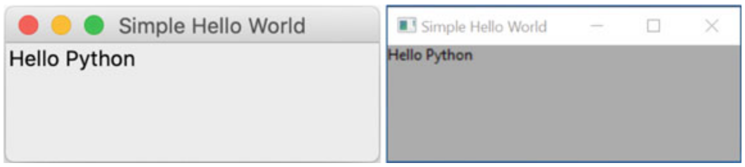
In wxPython, Frames and Dialogs as well as their contents are instances of appropriate classes (such as `Frame`, `Dialog`, `Panel`, `Button` or `StaticText`). Thus when you create a window, you create an object that knows how to display itself on the computer screen. You must tell it what to display and then tell it to show its contents to the user.

You should bear the following points in mind during your reading of this chapter; they will help you understand what you are required to do:

- You create a window by instantiating a Frame or Dialog object.
- You define what the window displays by creating a widget that has an appropriate parent component. This adds the widget to a container, such as a type of panel or a frame.
- You can send messages to the window to change its state, perform an operation, and display a graphic object.
- The window, or components within the window, can send messages to other objects in response to user (or program) actions.
- Everything displayed by a window is an instance of a class and is potentially subject to all of the above.
- `wx.App` handles the main event loop of the GUI application.

8.1.3 A Simple Example

An example of creating a very simple window using wxPython is given below. The result of running this short program is shown here for both a Mac and a Windows PC:



Hello Python

Hello Python

This program creates a top level window (the `wx.Frame`) and gives it a title. It also creates a label (a `wx.StaticText` object) to be displayed within the frame.

To use the wxPython library it is necessary to import the `wx` module.

```
import wx

# Create the Application Object
app = wx.App()
# Now create a Frame (representing the window)
frame = wx.Frame(parent=None, title='Simple Hello World')
# And add a text label to it
text = wx.StaticText(parent=frame, label='Hello Python')

# Display the window (frame)
frame.Show()

# Start the event loop
app.MainLoop()
```

The program also creates a new instance of the Application Object called `wx.App()`.

Every wxPython GUI program must have one Application Object. It is the equivalent of the `main()` function in many non-GUI applications as it will run the GUI application for you. It also provides default facilities for defining *startup* and *shutdown* operations and can be subclassed to create custom behaviour.

The `wx.StaticText` class is used to create a single (or multiple) line label. In this case the label shows the string 'Hello Python'. The `StaticText` object is constructed with reference to its parent container. This is the container within which the text will be displayed. In this case the `StaticText` is being displayed directly within the Frame and thus the frame object is its containing parent object. In contrast the Frame which is a top level window, does not have a parent container.

Also notice that the frame must be shown (displayed) for the user to see it. This is because there might be multiple different windows that need to be shown (or hidden) in different situations for an application.

Finally the program starts the applications' *main event* loop; within this loop the program listens for any user input (such as requesting that the window is closed).

8.2 The wx.App Class

The `wx.App` class represents the application and is used to:

- start up the wxPython system and initialise the underlying GUI toolkit,
- set and get application-wide properties,
- implement the native windowing system main message or event loop, and to dispatch events to window instances.

Every wxPython application must have a single `wx.App` instance. The creation of all of the UI objects should be delayed until after the `wx.App` object has been created in order to ensure that the GUI platform and wxWidgets have been fully initialised.

It is common to subclass the `wx.App` class and override methods such as `OnPreInit` and `OnExit` to provide custom behaviour. This ensures that the required behaviour is run at appropriate times. The methods that can be overridden for this purpose are:

- `OnPreInit`, This method can be overridden to define behaviour that should be run once the application object is created, but before the OnInit method has been called.
- `OnInit` This is expected to create the applications main window, display that window etc.
- `OnRun`, This is the method used to start the execution of the main program.
- `OnExit`, This can be overridden to provide any behaviour that should be called just before the application exits.

As an example, if we wish to set up a GUI application such that the main frame is initialised and shown after the `wx.App` has been instantiated then the safest way is to override the `OnInit()` method of the `wx.App` class in a suitable subclass. The method should return `True` of `False`; where `True` is used to indicate that processing of the application should continue and `False` indicates that the application should terminate immediately (usually as the result of some unexpected issue).

An example `wx.App` subclass is shown below:

```
class MainApp(wx.App):

    def OnInit(self):
        """ Initialise the main GUI Application"""
        frame = WelcomeFrame()
        frame.Show()
        # Indicate whether processing should continue or not
        return True
```

This class can now be instantiated and the `MainLoop` started, for example:

```
# Run the GUI application
app = MainApp()
app.MainLoop()
```

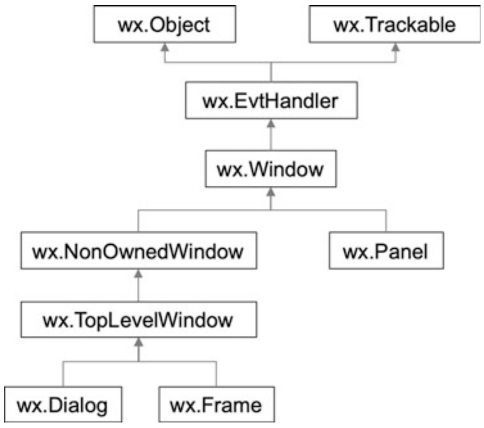
It is also possible to override the `OnExit()` to clean up anything initialised in the `OnInit()` method.

8.3 Window Classes

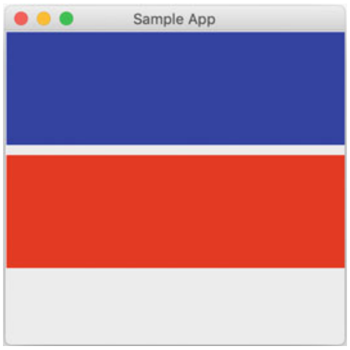
The window or widget container classes that are commonly used within a wxPython application are:

- `wx.Dialog` A Dialog is a top level window used for popups where the user has limited ability to interact with the window. In many cases the user can only input some data and/or accept or decline an option.
- `wx.Frame` A Frame is a top level window whose size and position can be set and can (usually) be controlled by the user.
- `wx.Panel` Is a container (non top level window) on which controls/widgets can be placed. This is often used in conjunction with a Dialog or a Frame to manage the positioning of widgets within the GUI.

The inheritance hierarchy for these classes is given below for reference:



As an example of using a `Frame` and a `Panel`, the following application creates two Panels and displays them within a top level `Frame`. The background colour of the Frame is the default grey; while the background colour for the first `Panel` is blue and for the second `Panel` it is red. The resulting display is shown below:



The program that generated this GUI is given below:

```
import wx

class SampleFrame(wx.Frame):

    def __init__(self):
        super().__init__(parent=None,
                         title='Sample App',
                         size=(300, 300))

        # Set up the first Panel to be at position 1, 1
        # (The default) and of size 300 by 100
        # with a blue background
        self.panel1 = wx.Panel(self)
        self.panel1.SetSize(300, 100)
        self.panel1.SetBackgroundColour(wx.Colour(0, 0, 255))

        # Set up the second Panel to be at position 1, 110
        # and of size 300 by 100 with a red background
        self.panel2 = wx.Panel(self)
        self.panel2.SetSize(1, 110, 300, 100)
        self.panel2.SetBackgroundColour(wx.Colour(255, 0, 0))

class MainApp(wx.App):

    def OnInit(self):
        """ Initialise the main GUI Application"""
        frame = SampleFrame()
        frame.Show()
        return True

# Run the GUI application
app = MainApp()
app.MainLoop()
```

The `SampleFrame` is a subclass of the `wx.Frame` class; it thus inherits all of the functionality of a Top Level Frame (window). Within the `__init__()` method of the `SampleFrame` the super classes `__init__()` method is called. This is used to set the size of the Frame and to give the Frame a title. Note that the Frame also indicates that it does not have a parent window.

When the Panel is created it is necessary to specify the window (or in this case Frame) within which it will be displayed. This is a common pattern within wxPython.

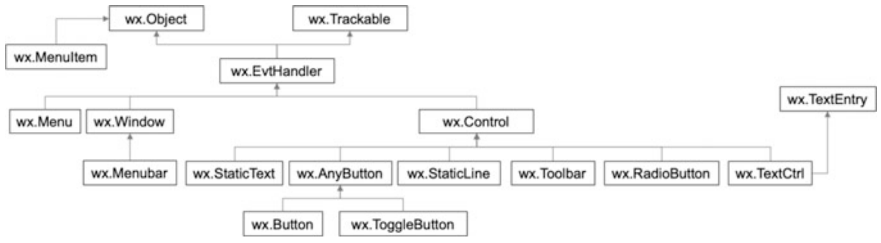
Also note that the `SetSize` method of the `Panel` class also allows the position to be specified and that the `Colour` class is the wxPython `Colour` class.

8.4 Widget/Control Classes

Although there are very many widgets/controls available to the developer, the most commonly used include:

- `wx.Button`/`wx.ToggleButton`/`wx.RadioButton` These are widgets that provide button like behaviour within a GUI.
- `wx.TextCtrl` This widget allows text to be displayed and edited. I can be a single line or multiple line widget depending upon configuration.
- `wx.StaticText` Used to display one or more lines of read-only text. In many libraries this widgets is known as a label.
- `wx.StaticLine` A line used in dialogs to separate groups of widgets. The line may be vertical or horizontal.
- `wx.ListBox` This widget is used to allow a user to select one option from a list of options.
- `wx.MenuBar`/`wx.Menu`/`wx.MenuItem`. The components that can be used to construct a set of menus for a User Interface.
- `wx.ToolBar` This widget is used to display a bar of buttons and/or other widgets usually placed below the menubar in a `wx.Frame`.

The inheritance hierarchy of these widgets is given below. Note that they all inherit from the class `Control` (hence why they are often referred to as Controls as well as Widgets or GUI components).



Whenever a widget is created it is necessary to provide the container window class that will hold it, such as a `Frame` or a `Panel`, for example:

```
enter_button = wx.Button(panel, label='Enter')
```

In this code snippet a `wx.Button` is being created that will have a label 'Enter' and will be displayed within the given Panel.

8.5 Dialogs

The generic `wx.Dialog` class can be used to build any custom dialog you require. It can be used to create *modal* and *modeless* dialogs:

- A modal dialog blocks program flow and user input on other windows until it is dismissed.
- A modeless dialog behaves more like a frame in that program flow continues, and input in other windows is still possible.
- The `wx.Dialog` class provides two versions of the show method to support modal and modeless dialogs. The `ShowModal()` method is used to display a modal dialog, while the `Show()` is used to show a modeless dialog.

As well as the generic wx.Dialog class, the wxPython library provides numerous prebuilt dialogs for common situations. These pre built dialogs include:

- `wx.ColourDialog` This class is used to generate a colour chooser dialog.
- `wx.DirDialog` This class provides a directory chooser dialog.
- `wx.FileDialog` This class provides a file chooser dialog.
- `wx.FontDialog` This class provides a font chooser dialog.
- `wx.MessageDialog` This class can be used to generate a single or multi-line message or information dialog. It can support Yes, No and Cancel options. It can be used for generic messages or for error messages.
- `wx.MultiChoiceDialog` This dialog can be used to display a lit of strings and allows the user to select one or more values for the list.
- `wx.PasswordEntryDialog` This class represents a dialog that allows a user to enter a one-line password string from the user.
- `wx.ProgressDialog` If supported by the GUI platform, then this class will provide the platforms native progress dialog, otherwise it will use the pure Python `wx.GenericProgressDialog`. The `wx.GenericProgressDialog` shows a short message and a progress bar.
- `wx.TextEntryDialog` This class provides a dialog that requests a one-line text string from the user.

Most of the dialogs that return a value follow the same pattern. This pattern returns a value from the `ShowModel()` method that indicates if the user selected OK or CANCEL (using the return value `wx.ID_OK` or `wx.ID_CANCEL`). The selected/entered value can then be obtained from a suitable get method such as `GetColourData()` for the `ColourDialog` or `GetPath()` for the `DirDialog`.

8.6 Arranging Widgets Within a Container

Widgets can be located within a window using specific coordinates (such as 10 pixels down and 5 pixels across). However, this can be a problem if you are considering cross platform applications, this is because how a button is rendered (drawn) on a Mac is different to Windows and different again from the windowing systems on Linux/Unix etc.

This means that different amount of spacing must be given on different platforms. In addition the fonts used with text boxes and labels differ between different platforms also requiring differences in the layout of widgets.

To overcome this wxPython provides *Sizers*. Sizers work with a container such as a Frame or a Panel to determine how the contained widgets should be laid out. Widgets are added to a sizer which is then set onto a container such as a Panel.

A Sizer is thus an object which works with a container and the host windowing platform to determine the best way to display the objects in the window. The developer does not need to worry about what happens if a user resizes a window or if the program is executed on a different windowing platform.

Sizers therefore help to produce portable, presentable user interfaces. In fact one Sizer can be placed within another Sizer to create complex component layouts.

There are several sizers available including:

- `wx.BoxSizer` This sizer can be used to place several widgets into a row or column organisation depending upon the orientation. When the BoxSizer is created the orientation can be specified using `wx.VERTICAL` or `wx, HORIZONTAL`.
- `wx.GridSizer` This sizer lays widgets out in a two dimensional grid. Each cell within the grid has the same size. When the `GridSizer` object is created it is possible to specify the number of rows and columns the grid has. It is also possible to specify the spacing between the cells both horizontally and vertically.
- `wx.FlexGridSizer` This sizer is a slightly more flexible version of the GridSizer. In this version not all columns and rows need to be the same size (although all cells in the same column are the same width and all cells in the same row are the same height).

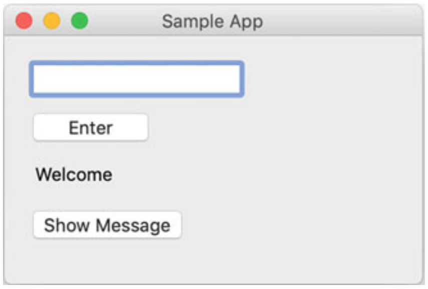
- `wx.GridBagSizer` is the most flexible sizer. It allows widgets to be positioned relative to the grid and also allows widgets to span multiple rows and/or columns.

To use a Sizer it must first be instantiated. When widgets are created they should be added to the sizer and then the sizer should be set on the container.

For example, the following code uses a `GridSizer` used with a `Panel` to layout out four widgets comprised of two buttons, a `StaticText` label and a `TextCtrl` input field:

```
# Create the panel
panel = wx.Panel(self)
# Create the sizer to use with 4 rows and 1 column
# And 5 spacing around each cell
grid = wx.GridSizer(4, 1, 5, 5)

# Create the widgets
text = wx.TextCtrl(panel, size=(150, -1))
enter_button = wx.Button(panel, label='Enter')
label = wx.StaticText(panel,label='Welcome')
message_button = wx.Button(panel, label='Show Message')

# Add the widgets to the grid sizer
grid.AddMany([text, enter_button, label, message_button])
# Set the sizer on the panel
panel.SetSizer(grid)
```

The resulting display is shown below:



8.7 Drawing Graphics

In earlier chapters we looked at the Turtle graphics API for generating vector and raster graphics in Python.

The wxPython library provides its own facilities for generating cross platform graphic displays using lines, squares, circles, text etc. This is provided via the Device Context.

A Device Context (often shortened to just DC) is an object on which graphics and text can be drawn.

It is intended to allow different output devices to all have a common graphics API (also known as the GDI or Graphics Device Interface). Specific device contexts can be instantiate depending on whether the program is to use a window on a computer screen or some other output medium (such as a printer).

There are several Device Context types available such as `wx.WindowDC`, `wx.PaintDC` and `wx.ClientDC`:

- The `wx.WindowDC` is used if we want to paint on the whole window (Windows only). This includes window decorations.
- The `wx.ClientDC` is used to draw on the client area of a window. The client area is the area of a window without its decorations (title and border).
- The `wx.PaintDC` is used to draw on the client area as well but is intended to support the window refresh paint event handling mechanism.

Note that the `wx.PaintDC` should be used only from a `wx.PaintEvent` handler while the `wx.ClientDC` should never be used from a `wx.PaintEvent` handler.

Whichever Device Context is used, they all support a similar set of methods that are used to generate graphics, such as:

- `DrawCircle (x, y, radius)` Draws a circle with the given centre and radius.
- `DrawEllipse (x, y, width, height)` Draws an ellipse contained in the rectangle specified either with the given top left corner and the given size or directly.
- `DrawPoint (x, y)` Draws a point using the color of the current pen.
- `DrawRectangle (x, y, width, height)` Draws a rectangle with the given corner coordinate and size.
- `DrawText (text, x, y)` Draws a text string at the specified point, using the current text font, and the current text foreground and background colours.
- `DrawLine (pt1, pt2)/DrawLine (x1, y1, x2, y2)` This method draws a line from the first point to the second.

It is also important to understand when the device context is refreshed/redrawn. For example, if you resize a window, maximise it, minimise it, move it, or modify its contents the window is redrawn. This generates an event, a `PaintEvent`.

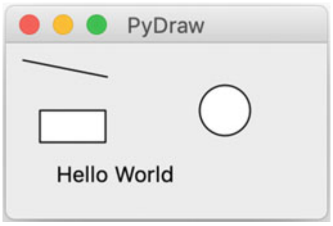
You can bind a method to the `PaintEvent` (using `wx.EVT_PAINT`) that can be called each time the window is refreshed.

This method can be used to draw whatever the contents of the window should be. If you do not redraw the contents of the device context in such a method than whatever you previously drew will display when the window is refreshed.

The following simple program illustrates the use of some of the Draw methods listed above and how a method can be bound to the paint event so that the display is refreshed appropriately when using a *device context*:

```
import wx

class DrawingFrame(wx.Frame):

    def __init__(self, title):
        super().__init__(None,
                         title=title,
                         size=(300, 200))

        self.Bind(wx.EVT_PAINT, self.on_paint)

    def on_paint(self, event):
        """set up the device context (DC) for painting"""
        dc = wx.PaintDC(self)
        dc.DrawLine(10, 10, 60, 20)
        dc.DrawRectangle(20, 40, 40, 20)
        dc.DrawText("Hello World", 30, 70)
        dc.DrawCircle(130, 40, radius=15)

class GraphicApp(wx.App):

    def OnInit(self):
        """ Initialise the GUI display"""
        frame = DrawingFrame(title='PyDraw')
        frame.Show()
        return True

# Run the GUI application
app = GraphicApp()
app.MainLoop()
```

When this program is run the following display is generated:



8.8 Online Resources

There are numerous online references that support the development of GUIs and of Python GUIs in particular, including:

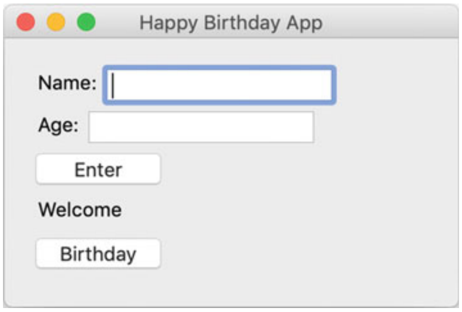
- https://docs.wxpython.org for documentation on wxPython.
- https://www.wxpython.org wxPython home page.
- https://www.wxwidgets.org For information on the underlying wxWidgets Cross platform GUI library.

8.9 Exercises

8.9.1 Simple GUI Application

In this exercise you will implement your own simple GUI application.

The application should generate the display for a simple UI. An example of the user interface is given below:



Notice that we have added a label to the input fields for name and age; you can manage their display using a nested panel.

In the next chapter we will add event handling to this application so that the application can respond to button clicks etc.