Chapter 9
Events in wxPython User Interfaces

9.1 Event Handling

Events are an integral part of any GUI; they represent user interactions with the interface such as clicking on a button, entering text into a field, selecting a menu option etc.

The main *event loop* listens for an event; when one occurs it processes that event (which usually results in a function or method being called) and then waits for the next event to happen. This loop is initiated in wxPython via the call to the `MainLoop()` method on the `wx.App` object.

This raises the question 'what is an Event?'. An event object is a piece of information representing some interaction that occurred typically with the GUI (although an event can be generated by anything). An event is processed by an *Event Handler*. This is a method or function that is called when the event occurs. The event is passed to the handler as a parameter. An *Event Binder* is used to bind an event to an event handler.

9.2 Event Definitions

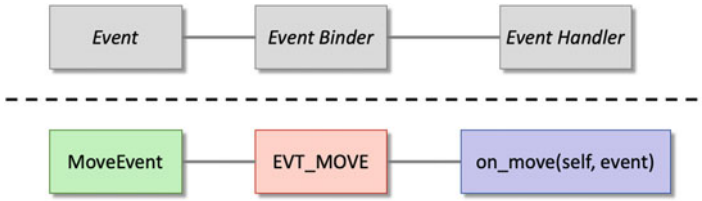
It is useful to summarise the definitions around events as the terminology used can be confusing and is very similar:

- **Event** represents information from the underlying GUI framework that describes something that has happened and any associated data. The specific data available will differ depending on what has occurred. For example, if a window has been moved then the associated data will relate to the window's new location. Where as a `CommandEvent` generated by a selection action from a ListBox provides the item index for the selection.

J. Hunt, *Advanced Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-25943-3_9

- **Event Loop** the main processing loop of the GUI that waits for an event to occur. When an event occurs the associated event handler is called.
- **Event Handlers** these are methods (or functions) that are called when an event occurs.
- **Event Binders** associate a type of event with an event handler. There are different event binders for different types of event. For example, the event binder associated with the `wx.MoveEvent` is named `wx.EVT_MOVE`.

The relationship between the Event, the Event Handler via the Event Binder is illustrated below:



The top three boxes illustrate the concepts while the lower 3 boxes provide a concrete example of binding a `Move_Event` to an `on_move()` method via the `EVT_MOVE` binder.

9.3 Types of Events

There are numerous different types of event including:

- `wx.CloseEvent` used to indicate that a Frame or Dialog has been closed. The event binder for this event is named `wx.EVT_CLOSE`.
- `wx.CommandEvent` used with widgets such as buttons, list boxes, menu items, radio buttons, scrollbars, sliders etc. Depending upon the type of widget that generated the event different information may be provided. For example, for a `Button` a `CommandEvent` indicates that a button has been clicked where as for a `ListBox` it indicates that an option has been selected, etc. Different event binders are used for different event situations. For example, to bind a command event to a event handler for a button then the `wx.EVT_BUTTON` binder is used; while for a ListBox a `wx.EVT_LISTBOX` binder can be used.
- `wx.FocusEvent` This event is sent when a window's focus changes (loses or gains focus). You can pick up a window gaining focus using the `wx.EVT_SET_FOCUS` event binder. The `wx.EVT_KILL_FOCUS` is used to bind an event handler that will be called when a window loses focus.

- `wx.KeyEvent` This event contains information relating to a key press or release.
- `wx.MaximizeEvent` This event is generated when a top level window is maximised.
- `wx.MenuEvent` This event is used for menu oriented actions such as the menu being opened or closed; however it should be noted that this event is not used when a menu item is selected (`MenuItems` generate `CommandEvents`).
- `wx.MouseEvent` This event class contains information about the events generated by the mouse: this includes information on which mouse button was pressed (and released) and whether the mouse was double clicked etc.
- `wx.WindowCreateEvent` This event is sent just after the actual window is created.
- `wx.WindowDestoryedEvent` This event is sent as early as possible during the window destruction process.

9.4 Binding an Event to an Event Handler

An event is bound to an Event Handler using the `Bind()` method of an event generating object (such as a button, field, menu item etc.) via a named Event Binder.

For example:

```
button.Bind(wx.EVT_BUTTON, self.event_handler_method)
```

9.5 Implementing Event Handling

There are four steps involved in implementing event handling for a widget or window, these are:

1. **Identify the event of interest**. Many widgets will generate different events in different situations; it may therefore be necessary to determine which event you are interested in.
2. **Find the correct Event Binder name**, e.g. wx.EVT_CLOSE, wx.EVT_MOVE or wx.EVT_BUTTON etc. Again you may find that the widget you are interested in supports numerous different event binders which may be used in different situations (even for the same event).
3. **Implement an event handler** (i.e. a suitable method or function) that will be called when the event occurs. The event handler will be supplied with the event object.
4. **Bind the Event to the Event Handler** via the Binder Name using the `Bind()` method of the widget or window.

To illustrate this we will use a simple example.

We will write a very simple event handling application. This application will have a `Frame` containing a `Panel`. The Panel will contain a label using the `wx.StaticText` class.

We will define an event handler called `on_mouse_click()` that will move the `StaticText` label to the current mouse location when the left mouse button is pressed. This means that we can move the label around the screen.

To do this we first need to determine the widget that will be used to generate the event. In this case it is the panel that contains the text label. Having done this we can look at the `Panel` class to see what events and Event Bindings it supports. It turns out that the `Panel` class only directly defines support for `NavigationKeyEvents`. This is not actually what we want; however the `Panel` class extends the `Window` class.

The `Window` class supports numerous event bindings, from those associated with setting the focus (`wx.EVT_SET_FOCUS` and `wx.EVT_KILL_FOCUS`) to key presses (`wx.EVT_KEY_DOWN` and `wx.EVT_KEY_UP`) as well as mouse events. There are however numerous different mouse event bindings. These allow left, middle and right mouse button clicks to be picked up, down clicks to be identified, situations such as the mouse entering or leaving the window etc. However, the binding we are interested in for a MouseEvent is the `wx.EVT_LEFT_DOWN` binding; this picks up on the `MoueEvent` when the left mouse button is pressed (there is also the `wx.EVT_LEFT_UP` binding which can be used to pick up an event that occurs when the left mouse button is released).

We now know that we need to bind the `on_mouse_click()` event handler to the MouseEvent via the `wx.EVT_LEFT_DOWN` event binder, for example:

```
self.panel.Bind(wx.EVT_LEFT_DOWN, self.on_mouse_click)
```

All event handler methods takes two parameters, `self` and the mouse event. Thus the signature of the `on_mouse_click()` method is:

```
def on_mouse_click(self, mouse_event):
```

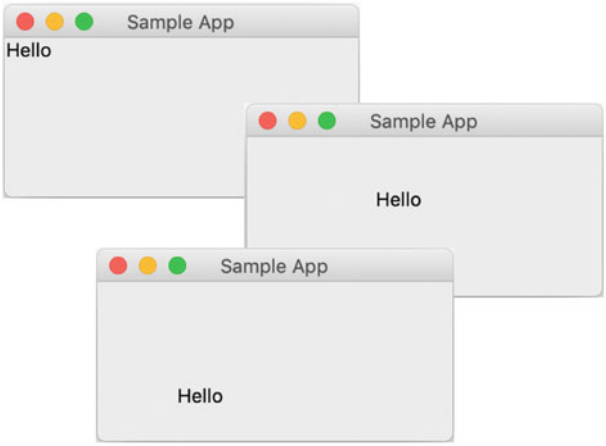
The mouse event object has numerous methods defined that allow information about the mouse to be obtained such as the number of mouse clicks involved (`GetClickCount()`), which button was pressed (`GetButton()`) and the current mouse position within the containing widget or window (`GetPosition()`). We can therefore use this last method to obtain the current mouse location and then use the `SetPosition(x, y)` method on the `StaticText` object to set its position.

The end result is the program shown below:

```
import wx

class WelcomeFrame(wx.Frame):
    """ The Main Window / Frame of the application """

    def __init__(self):
        super().__init__(parent=None,
                         title='Sample App',
                         size=(300, 200))

        # Set up panel within the frame and text label
        self.panel = wx.Panel(self)
        self.text = wx.StaticText(self.panel,
                                  label='Hello')

        # Bind the on_mouse_click method to the
        # Mouse Event via the
        # left mouse click binder
        self.panel.Bind(wx.EVT_LEFT_DOWN,
                        self.on_mouse_click)

    def on_mouse_click(self, mouse_event):
        """ When the left mouse button is clicked
            This method is called. It will obtain
            the current mouse coordinates, and
            reposition the text label
            to this position. """
        x, y = mouse_event.GetPosition()
        print(x, y)
        self.text.SetPosition(wx.Point(x, y))

class MainApp(wx.App):

    def OnInit(self):
        """ Initialise the main GUI Application"""
        frame = WelcomeFrame()
        frame.Show()
        # Indicate that processing should continue
        return True

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

When this program is run; the window is displayed with the 'Hello' `StaticText` label in the top left hand corner of the Frame (actually it is added to the Panel, however the Panel fills the `Frame` in this example). If the user then clicks the left mouse button anywhere within the Frame then the 'Hello' label jumps to that location.

This is shown below for the initial setup and then for two locations within the window.

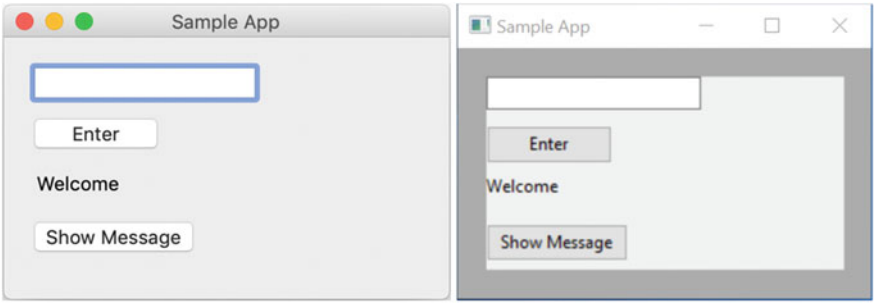


9.6 An Interactive wxPython GUI

An example of a slightly larger GUI application, that brings together many of the ideas presented in this chapter, is given below.

In this application we have a text input field (a `wx.TextCtrl`) that allows a user to enter their name. When they click on the Enter button (`wx.Button`) the welcome label (a `wx.StaticText`) is updated with their name. The 'Show Message' button is used to display a `wx.MessageDialog` which will also contain their name.

The initial display is shown below for both a Mac and a Windows PC, note that the default background colour for a Frame is different on a Windows PC than on a Mac and thus although the GUI runs on both platforms, the look differs between the two:



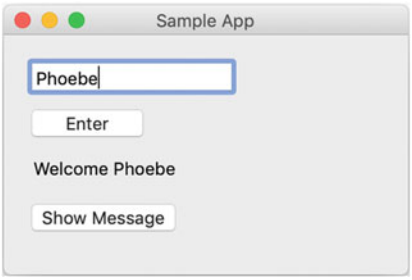
The code used to implement this GUI application is given below:

```
import wx

class HelloFrame(wx.Frame):

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

        self.name = '<unknown>'

        # Create the BoxSizer to use for the Frame
        vertical_box_sizer = wx.BoxSizer(wx.VERTICAL)
        self.SetSizer(vertical_box_sizer)

        # Create the panel to contain the widgets
        panel = wx.Panel(self)
        # Add the Panel to the Frames Sizer
        vertical_box_sizer.Add(panel,
                               wx.ID_ANY,
                               wx.EXPAND | wx.ALL,
                               20)

        # Create the GridSizer to use with the Panel
        grid = wx.GridSizer(4, 1, 5, 5)

        # Set up the input field
        self.text = wx.TextCtrl(panel, size=(150, -1))
```

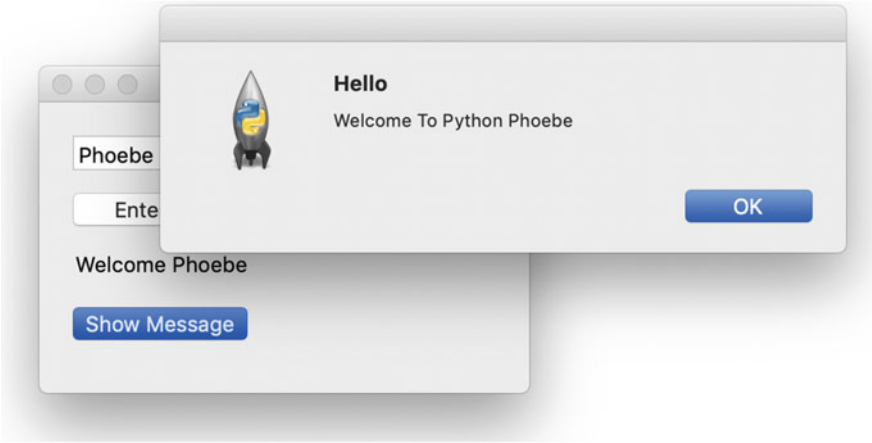
```
        # Now configure the enter button
        enter_button = wx.Button(panel, label='Enter')
        enter_button.Bind(wx.EVT_BUTTON, self.set_name)

        # Next set up the text label
        self.label = wx.StaticText(panel,
                                   label='Welcome',
                                   style=wx.ALIGN_LEFT)

        # Now configure the Show Message button
        message_button = wx.Button(panel, label='Show Message')
        message_button.Bind(wx.EVT_BUTTON, self.show_message)

        # Add the widgets to the grid sizer to handle layout
        grid.AddMany([self.text,
                      enter_button,
                      self.label,
                      message_button])

        # Set the sizer on the panel
        panel.SetSizer(grid)

        # Centre the Frame on the Computer Screen
        self.Centre()

    def show_message(self, event):
        """ Event Handler to display the Message Dialog
        using the current value of the name attribute. """
        dialog = wx.MessageDialog(None,
                  message='Welcome To Python ' + self.name,
                  caption='Hello',
                  style=wx.OK)
        dialog.ShowModal()

    def set_name(self, event):
        """ Event Handler for the Enter button.
            Retrieves the text entered into the input field
            and sets the self.name attribute. This is then
            used to set the text label """
        self.name = self.text.GetLineText(0)
        self.label.SetLabelText('Welcome ' + self.name)
```

```
class MainApp(wx.App):

    def OnInit(self):
        """ Initialise the GUI display"""
        frame = HelloFrame(title='Sample App')
        frame.Show()
        # Indicate whether processing should continue or not
        return True

    def OnExit(self):
        """ Executes when the GUI application shuts down"""
        print('Goodbye')
        # Need to indicate success or failure
        return True

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

If the user enters their name in the top TextCtrl field, for example 'Phoebe', then when they click on the 'Enter' button the welcome label changes to 'Welcome Phoebe':



If they now click on the 'Show Message' button then the `wx.MessageDialog` (a specific type of `wx.Dialog`) will display a welcome message to Phoebe:



9.7 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.

9.8 Exercises

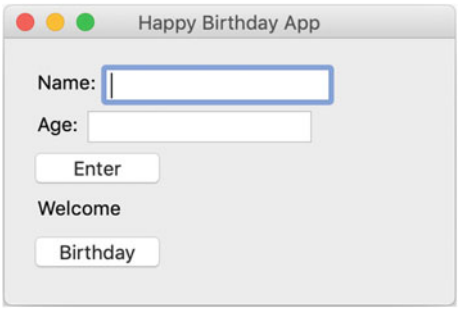
9.8.1 Simple GUI Application

This exercise builds on the GUI you created in the last chapter.

The application should allow a user to enter their name and age. You will need to check that the value entered into the age field is a numeric value (for example using `isnumeric()`). If the value is not a number then an error message dialog should be displayed.

A button should be provided labelled ‘Birthday’; when clicked it should increment the age by one and display a Happy Birthday message. The age should be updated within the GUI.

An example of the user interface you created in the last chapter is given below:



As an example, the user might enter their name and age as shown below:

When the user clicks on the ‘birthday’ button then the Happy Birthday message dialog is displayed:

9.8.2 GUI Interface to a Tic Tac Toe Game

The aim of this exercise is to implement a simple Tic Tac Toe game. The game should allow two users to play interactive using the same mouse. The first user will have play as the 'X' player and the second user as the '0' player.

When each user selects a button you can set the label for the button to their symbol.

You will need two check after each move to see if someone has won (or if the game is a draw).

You will still need an internal representation of the grid so that you can determine who, if anyone, has won.

An example of how the GUI for the TicTacToe game might look is given below:

You can also add dialogs to obtain the players names and to notify them who won or whether there was a draw.