

Advanced wxPython tutorial

Jan Bodnar

November 13, 2021

Contents

0.1	About the author	4
0.2	About the e-book	5
I	Core	6
1	Layout management	7
1.1	Absolute positioning	7
1.2	Box sizers	9
1.2.1	A row of buttons	10
1.2.2	Nesting	10
1.2.3	Gaps	12
1.2.4	Proportions	13
1.2.5	Proportions 2	14
1.2.6	Alignments	15
1.2.7	Buttons example	16
1.2.8	New folder example	18
1.2.9	Windows example	20
1.3	wx.GridBagSizer	22
1.3.1	Simple example	22
1.3.2	Gaps	23
1.3.3	Spanning	24
1.3.4	Spanning 2	25
1.3.5	Alignments	26
1.3.6	Alignments 2	27
1.3.7	Growable rows and columns	28
1.3.8	New folder example	30
1.3.9	Windows example	32
2	Images	35
2.1	Blurring image	35
2.2	Scaling image	36
2.3	Cropping image	38
2.4	Embedded image	39
2.5	Art provider	40
2.6	Grayscale image	42
2.7	Watermark	45
2.8	Screenshot	46

3	Custom widgets	50
3.1	ProgressMeter widget	50
3.2	Thermometer widget	54
3.3	ColourWheel widget	62
II	Advanced widgets	75
4	wx.TextCtrl	76
4.1	Simple example	76
4.2	Lines and columns	79
4.3	Selections	82
4.4	Text search example	84
4.5	Load, save example	87
5	wx.ListCtrl	96
5.1	Simple example in a report view	96
5.2	Selections	99
5.3	Editable list control	102
5.4	Background color & auto width mixin	104
5.5	Images in a report view	106
5.6	Icon view	108
5.7	List control with check boxes	111
5.8	Sortable columns	114
5.9	Virtual list control	118
6	wx.TreeCtrl	122
6.1	Simple example	122
6.2	Selections	124
6.3	Traversing a tree control	127
6.4	Searching	130
6.5	Sorting items	134
6.6	Images	139
6.7	Lazy tree evaluation	141
7	wx.grid.Grid	145
7.1	Basics	145
7.2	Cell attributes	147
7.3	Show hide example	148
7.4	Moving the cursor	151
7.5	Enter key	153
7.6	Selections	155
7.7	Merging cells	159
7.8	Cell editors	163
7.9	Cell renderers	164
7.10	Custom cell renderer	165
7.11	Model & view	168
7.12	Model & view II	171

8	wx.richtext.RichTextCtrl	176
8.1	Introductory example	176
8.2	Methods	177
8.3	Bullets	179
8.4	Images and URL links	181
8.5	Font weight and style	184
8.6	Undo and redo	187
8.7	Saving in XML	189
8.8	Loading from XML	191
III	Games	194
9	Snake	195
10	Sokoban	205
11	Minesweeper	219

0.1 About the author

My name is Jan Bodnar. I am Hungarian and I come from Slovakia. Currently I live in Bratislava. Apart from working with computers I read a lot. Mostly belles-letters and history. I study several foreign languages. Being aware of that healthy soul lives in a healthy body, I exercise regularly, work in the garden and often take tours.

I run the zetcode.com site. I have also written the following e-books: *Swing layout management*, *Advanced PyQt5*, *Tkinter programming*, *Introduction to Windows API*, *SQLite Python*, and *Advanced Java Swing*. I hope my e-book will be useful to you.



Thank you for buying this e-book.

0.2 About the e-book

This is Advanced wxPython tutorial. This tutorial covers several parts of the wxPython library in a great detail.

This e-book covers the following:

- Cairo
- Layout management
- Images
- Custom widgets
- Advanced widgets
- Snake game
- Sokoban game
- Minesweeper game

In the first chapter, we explore the Cairo 2D library. We work with colours, shapes, gradients. Reflection, star animation, aliens example are among other code examples. The Cairo library will be later used to create custom widgets and games.

In the Layout management chapter, we use built-in layout management classes to lay out our widgets on the window.

In the Images chapter, we work with images. We show how to scale or crop the image. We create a grayscale image, a watermark or a an application which takes a screenshot.

In the Custom widgets part, we show how to create custom widgets. We create a ProgressMeter widget, a Thermometer widget, and a ColorWheel widget.

The fifth chapter covers advanced widgets, including the text control, list control, tree control, grid control, and rich text control.

One of the best ways to study programming is to create simple computer games. In this e-book, we create three classic 2D games: Snake, Sokoban, and Minesweeper.

In all chapters, you will find lots of practical examples. They are additional material to the wxPython tutorial, which can be found on the zetcode.com website.

The official wxPython documentation and the wxPython demo application were consulted while creating this e-book.

Part I

Core

Chapter 1

Layout management

In this chapter, we cover the layout management in wxPython. When we design the GUI of our application, we decide what widgets we use and how we organize those widgets in the application. To organize our widgets, we use specialized non-visible objects called layout managers.

Layout managers are software components used in widget toolkits which have the ability to lay out widgets by their relative positions without using distance units. It is often more natural to define component layouts in this manner than to define their position in pixels or common distance units.

To create our layouts, we use two principal layout managers: the `wx.BoxSizer` and the `wx.GridBagSizer`. The `wx.BoxSizer` is a simple layout manager that can line up widgets either vertically or horizontally. With the help of nesting, we can produce quite complicated layouts with box sizers. The `wx.GridBagSizer` is the most complex layout manager which is capable of creating the most complicated layouts.

1.1 Absolute positioning

In absolute positioning, the programmer specifies the position and the size of each component in pixels. We have to understand several things:

- The size and the position of a component do not change, if we resize a window.
- Applications might look different on various platforms.
- Changing fonts in our application might spoil the layout.
- If we decide to change our layout, we must completely redo our layout, which is tedious and time consuming.

In most applications, we don't use absolute positioning. However, there are some specialized areas, where we can use it. For example, games, specialized applications that work with diagrams, resizable components that can be moved (like a chart in a spreadsheet application), small educational examples.

In the following example, we show three images. They are placed on the window using absolute positioning. Images are loaded from the disk. They are placed inside `wx.StaticBitmap` widgets.

Listing 1.1: Absolute positioning

```
def InitUI(self):  
  
    self.SetBackgroundColour(wx.Colour(30, 30, 30))  
  
    wx.StaticBitmap(self, bitmap=wx.Bitmap("bar.jpg"),  
                    pos=(20, 20))  
    wx.StaticBitmap(self, bitmap=wx.Bitmap("min.jpg"),  
                    pos=(40, 160))  
    wx.StaticBitmap(self, bitmap=wx.Bitmap("rot.jpg"),  
                    pos=(170, 50))
```

In the code example, we show three images on the window. If we resize the window, the position and the size of the images do not change.

```
self.SetBackgroundColour(wx.Colour(30, 30, 30))
```

We set the background of the window to dark gray colour.

```
wx.StaticBitmap(self, bitmap=wx.Bitmap("bar.jpg"),  
                pos=(20, 20))
```

We create a `wx.StaticBitmap` widget. It takes three parameters: the parent widget, the bitmap, and the position of the bitmap on the window.

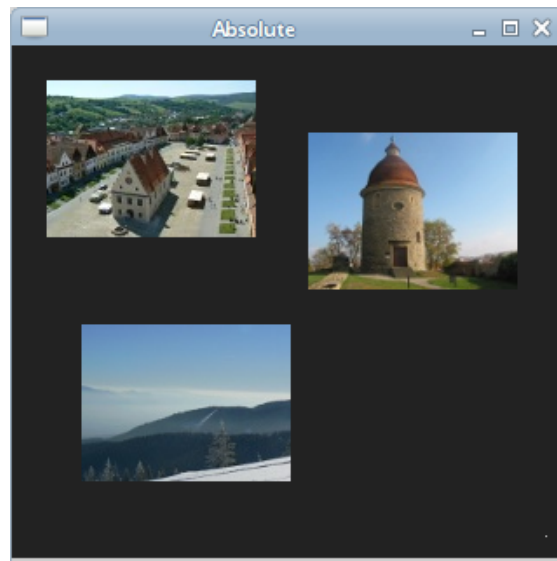


Figure 1.1: Absolute positioning

Figure 1.1 shows three pictures placed on the window using absolute positioning.

1.2 Box sizers

A box sizer is created with a `wx.BoxSizer` class. Box sizers are very simple layout managers. They line up child widgets horizontally or vertically. The constructor takes a single `orient` parameter. To create a vertical box sizer, we provide the `wx.VERTICAL` value. A horizontal box sizer is created with the `wx.HORIZONTAL` value.

Widgets and other box sizers are added to a box sizer using the `Add` method. The first parameter of the method is the widget which is being added. There are three optional parameters. If they are not specified, they take default values. The `proportion` parameter defines the ratio of how widgets change in the given orientation. A widget with proportion 0 does not change. A widget with proportion 2 grows or shrinks horizontally twice as much as a widget with proportion 1 located in a horizontal box sizer.

The `flag` parameter has multiple functions. It does align widgets in the box sizers with align flags. For example, a widget with `wx.ALIGN_LEFT` is aligned to the left in a horizontal box sizer. A widget with `wx.ALIGN_BOTTOM` is aligned to the bottom in a vertical box sizer.

The following list shows all flags that can be used to align widgets in box sizers.

- `wx.ALIGN_CENTER`
- `wx.ALIGN_LEFT`
- `wx.ALIGN_RIGHT`
- `wx.ALIGN_TOP`
- `wx.ALIGN_BOTTOM`
- `wx.ALIGN_CENTER_VERTICAL`
- `wx.ALIGN_CENTER_HORIZONTAL`

The `wx.EXPAND` flag causes the widget to take all the space available in the box sizer. A button with this flag (and a proportion set to 1) in a horizontal box sizer will be stretched from the left to the right.

Finally, there is a `border` parameter. It adds some border/space to the sides of the widget. It is used in combination with the `flag` parameter, which provides the sides, where the space will be added. For example, if we have a border and a `wx.TOP` flag specified, the space will be added to the top of the widget.

The following list shows all flags that are used in combination with the border parameter.

- `wx.TOP`
- `wx.BOTTOM`
- `wx.RIGHT`
- `wx.LEFT`
- `wx.ALL`

1.2.1 A row of buttons

In the following example, we create a row of 3 buttons.

Listing 1.2: A row of buttons

```
def InitUI(self):  
  
    pnl = wx.Panel(self)  
  
    hbox = wx.BoxSizer(wx.HORIZONTAL)  
  
    hbox.Add(wx.Button(pnl, label="Button"))  
    hbox.Add(wx.Button(pnl, label="Button"))  
    hbox.Add(wx.Button(pnl, label="Button"))  
  
    pnl.SetSizer(hbox)
```

We use a `wx.BoxSizer` to create a row of three buttons.

```
hbox = wx.BoxSizer(wx.HORIZONTAL)
```

In this code line, a horizontal box sizer is created.

```
hbox.Add(wx.Button(pnl, label="Button"))  
hbox.Add(wx.Button(pnl, label="Button"))  
hbox.Add(wx.Button(pnl, label="Button"))
```

Three buttons are added to the horizontal box. The other parameters of the `Add` method are not specified; they take default values. There is no border between the buttons. The buttons do not change their size when the window is resized.

```
pnl.SetSizer(hbox)
```

We set the sizer for the panel widget. A panel can take only one layout manager. (A sizer is a synonym for a layout manager.)

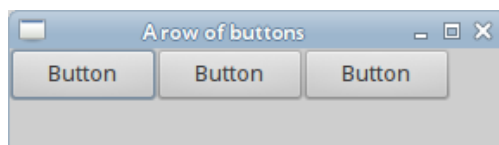


Figure 1.2: A row of buttons

Figure 1.2 shows three buttons placed in one horizontal row.

1.2.2 Nesting

It is possible to nest box sizers. This way we can create complicated layouts. We can put horizontal box sizers into vertical box sizers and vice versa.

Listing 1.3: Nesting of box sizers

```
def InitUI(self):
```

```

pnl = wx.Panel(self)

hbox = wx.BoxSizer(wx.HORIZONTAL)
pnl.SetSizer(hbox)

vbox1 = wx.BoxSizer(wx.VERTICAL)
vbox2 = wx.BoxSizer(wx.VERTICAL)
vbox3 = wx.BoxSizer(wx.VERTICAL)
vbox4 = wx.BoxSizer(wx.VERTICAL)

vbox1.Add(wx.Button(pnl, label="Button"))
vbox2.Add(wx.Button(pnl, label="Button"))
vbox2.Add(wx.Button(pnl, label="Button"))
vbox3.Add(wx.Button(pnl, label="Button"))
vbox3.Add(wx.Button(pnl, label="Button"))
vbox4.Add(wx.Button(pnl, label="Button"))
vbox4.Add(wx.Button(pnl, label="Button"))
vbox4.Add(wx.Button(pnl, label="Button"))

hbox.Add(vbox1, flag=wx.ALIGN_CENTER_VERTICAL)
hbox.Add(vbox2, flag=wx.ALIGN_CENTER_VERTICAL)
hbox.Add(vbox3, flag=wx.ALIGN_CENTER_VERTICAL)
hbox.Add(vbox4, flag=wx.ALIGN_CENTER_VERTICAL)

```

In our example, we have one horizontal box sizer. We nest four vertical boxes into this horizontal box sizer. We place buttons into the vertical boxes. The buttons are aligned vertically. The alignment will be explained later.

```

hbox = wx.BoxSizer(wx.HORIZONTAL)
pnl.SetSizer(hbox)

```

We create a base horizontal box sizer. We nest all vertical box sizers into this horizontal box sizer.

```

vbox1 = wx.BoxSizer(wx.VERTICAL)
vbox2 = wx.BoxSizer(wx.VERTICAL)
vbox3 = wx.BoxSizer(wx.VERTICAL)
vbox4 = wx.BoxSizer(wx.VERTICAL)

```

Here we create four vertical box sizers.

```

vbox1.Add(wx.Button(pnl, label="Button"))
vbox2.Add(wx.Button(pnl, label="Button"))
vbox2.Add(wx.Button(pnl, label="Button"))
...

```

One button is added to the first vertical box sizer and two buttons into the second one.

```

hbox.Add(vbox1, flag=wx.ALIGN_CENTER_VERTICAL)
hbox.Add(vbox2, flag=wx.ALIGN_CENTER_VERTICAL)
hbox.Add(vbox3, flag=wx.ALIGN_CENTER_VERTICAL)
hbox.Add(vbox4, flag=wx.ALIGN_CENTER_VERTICAL)

```

The vertical box sizers are added to the horizontal box sizer. The buttons are

vertically centered.

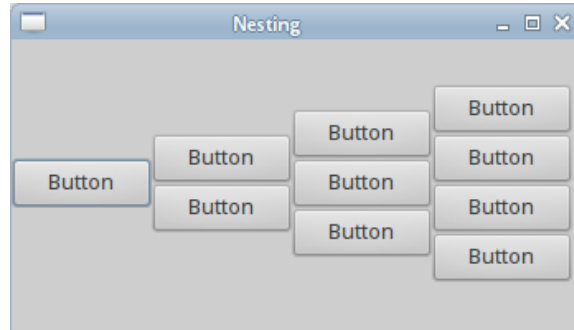


Figure 1.3: Nesting

Figure 1.3 shows buttons lined up vertically. To achieve this layout, we have used vertical box sizers nested into one horizontal box sizer.

1.2.3 Gaps

The `border` parameter puts space around a widget. In the `flag` parameter, we specify where the border space is applied.

Listing 1.4: Gaps

```
def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    hbox1 = wx.BoxSizer(wx.HORIZONTAL)
    hbox2 = wx.BoxSizer(wx.HORIZONTAL)

    hbox1.Add(wx.Button(pnl, label="Button"),
               flag=wx.LEFT | wx.RIGHT, border=3)
    hbox1.Add(wx.Button(pnl, label="Button"),
               flag=wx.RIGHT, border=3)
    hbox1.Add(wx.Button(pnl, label="Button"),
               flag=wx.RIGHT, border=3)
    hbox2.Add(wx.Button(pnl, label="Button"),
               flag=wx.LEFT | wx.RIGHT, border=3)
    hbox2.Add(wx.Button(pnl, label="Button"),
               flag=wx.RIGHT, border=3)
    hbox2.Add(wx.Button(pnl, label="Button"),
               flag=wx.RIGHT, border=3)

    vbox.Add(hbox1, flag=wx.TOP | wx.BOTTOM,
              border=5)
    vbox.Add(hbox2, flag=wx.BOTTOM, border=5)

    pnl.SetSizer(vbox)
```

We have two rows of three buttons. We put some space between the buttons, between the rows of buttons, and between the rows of buttons and the window borders.

```
hbox1.Add(wx.Button(pnl, label="Button"),
          flag=wx.LEFT | wx.RIGHT, border=3)
```

We add a button widget to the horizontal box sizer. We put a space of three units to the left and the right of the button.

```
vbox.Add(hbox1, flag=wx.TOP | wx.BOTTOM,
         border=5)
```

The first horizontal box sizer is added to the base vertical box sizer. Also a border is created above and below the sizer. As a result, the first row of buttons is separated from the top border of the window and from the second row of buttons.



Figure 1.4: Gaps

In Figure 1.4 we see gaps between button widgets on the window.

1.2.4 Proportions

The proportion parameter defines the ratio of the change of a widget when a window is resized. A widget with proportion 0 does not change at all. A widget with a proportion 2 changes twice as much as the one with proportion 1 in the given dimension. A proportion parameter is used in combination with the `wx.EXPAND` flag. A widget may or may not take all the space that is available in the box sizer.

Listing 1.5: Proportions

```
def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    hbox1 = wx.BoxSizer(wx.HORIZONTAL)
    hbox2 = wx.BoxSizer(wx.HORIZONTAL)
    hbox3 = wx.BoxSizer(wx.HORIZONTAL)

    hbox1.Add(wx.Button(pnl, label="Button"),
              proportion=0)
    hbox2.Add(wx.Button(pnl, label="Button"),
              proportion=1)
    hbox3.Add(wx.Button(pnl, label="Button"),
              proportion=1)
```

```

vbox.Add(hbox1, flag=wx.EXPAND)
vbox.Add(hbox2, flag=wx.EXPAND)
vbox.Add(hbox3)

pnl.SetSizer(vbox)

```

There are three buttons in three horizontal box sizers. We combine the proportion parameter with the `wx.EXPAND` flag.

```

hbox1.Add(wx.Button(pnl, label="Button"),
          proportion=0)
...
vbox.Add(hbox1, flag=wx.EXPAND)

```

A button with a proportion 0 does not change and it retains its default size. The horizontal box sizer with the `wx.EXPAND` flag is stretched horizontally. The button does not take all the space that is available in the box sizer.

```

hbox2.Add(wx.Button(pnl, label="Button"),
          proportion=1)
...
vbox.Add(hbox2, flag=wx.EXPAND)

```

In the second case, we have a button with a proportion 1. It is added to the expanded horizontal box sizer. The button is stretched horizontally.

```

hbox3.Add(wx.Button(pnl, label="Button"),
          proportion=1)
...
vbox.Add(hbox3)

```

Finally, a button with a proportion 1 is added to a non-expanded horizontal box sizer. The button retains its initial size. It has no room to expand.

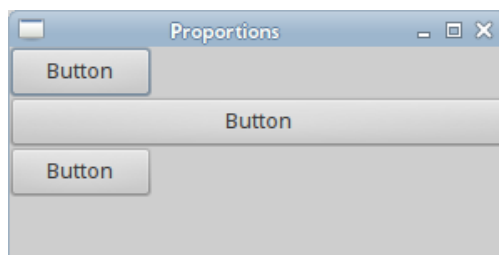


Figure 1.5: Proportions

Figure 1.5 shows the first proportions example. One of the three buttons is stretched horizontally.

1.2.5 Proportions 2

In the second example, we show widgets that have different proportion values.

Listing 1.6: Proportions 2

```
def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    hbox1 = wx.BoxSizer(wx.HORIZONTAL)

    hbox1.Add(wx.Button(pnl, label="Button"),
               proportion=0)
    hbox1.Add(wx.Button(pnl, label="Button"),
               proportion=1)
    hbox1.Add(wx.Button(pnl, label="Button"),
               proportion=2)

    vbox.Add(hbox1, flag=wx.EXPAND)

    pnl.SetSizer(vbox)
```

There are three buttons in a horizontal box sizer. The first button does not change at all. The button with proportion 2 changes horizontally twice as much as the button with proportion 1.

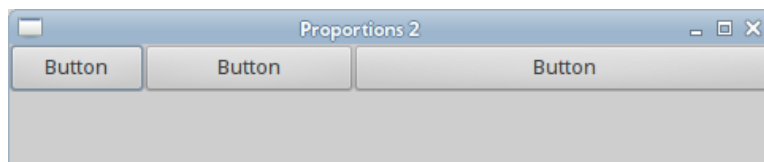


Figure 1.6: Proportions 2

Figure 1.6 shows the second proportions example. We can see that the buttons have different width.

1.2.6 Alignments

Within layout managers, it is possible to align widgets. In a vertical box, we can align widgets to the top, center, or to the bottom. In a horizontal box, we can align widgets to the left, center, or to the right. The alignment is controlled with a `flag` parameter.

Listing 1.7: Horizontal alignment of widgets

```
def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    hbox1 = wx.BoxSizer(wx.HORIZONTAL)
    hbox2 = wx.BoxSizer(wx.HORIZONTAL)
    hbox3 = wx.BoxSizer(wx.HORIZONTAL)
    hbox4 = wx.BoxSizer(wx.HORIZONTAL)

    hbox1.Add(wx.Button(pnl, label="Left"))
    hbox2.Add(wx.Button(pnl, label="Center"))
    hbox3.Add(wx.Button(pnl, label="Right"))
    hbox4.Add(wx.Button(pnl, label="Stretch"),
```



```

        proportion=1)

    vbox.Add(hbox1, flag=wx.ALIGN_LEFT)
    vbox.Add(hbox2, flag=wx.ALIGN_CENTER)
    vbox.Add(hbox3, flag=wx.ALIGN_RIGHT)
    vbox.Add(hbox4, flag=wx.EXPAND)

    pnl.SetSizer(vbox)

```

In our example, we have four horizontal boxes nested in one vertical box. In these four horizontal boxes, we align button widgets to the left, center and to the right. The last button is stretched from the left to the right.

```
vbox.Add(hbox1, flag=wx.ALIGN_LEFT)
```

The `wx.ALIGN_LEFT` flag aligns the horizontal box with the button to the left.

```
vbox.Add(hbox4, flag=wx.EXPAND)
```

The fourth horizontal box is expanded. The button has proportion 1. Therefore, it is stretched horizontally.

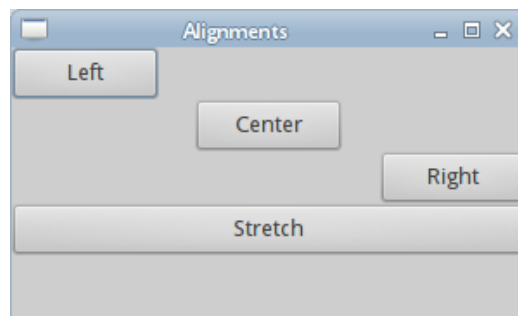


Figure 1.7: Alignments

Figure 1.7 shows three buttons aligned horizontally and one button stretched from the left to the right.

1.2.7 Buttons example

We have our first practical example. We put two buttons in the bottom right corner of the window.

Listing 1.8: Buttons example

```

def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    pnl.SetSizer(vbox)

    hbox = wx.BoxSizer(wx.HORIZONTAL)

```

```

okButton = wx.Button(pnl, label="OK")
cancelButton = wx.Button(pnl, label="Cancel")

hbox.Add(okButton, flag=wx.RIGHT | wx.LEFT,
         border=5)
hbox.Add(cancelButton, flag=wx.RIGHT,
         border=5)

vbox.InsertStretchSpacer(0)
vbox.Add(hbox, flag=wx.ALIGN_RIGHT|wx.BOTTOM,
         border=5)

```

When we resize the window, the buttons stay in the corner. To achieve this layout, we use one vertical box sizer and one horizontal box sizer.

```

vbox = wx.BoxSizer(wx.VERTICAL)
pnl.SetSizer(vbox)

```

The panel's base sizer is a vertical box sizer.

```

hbox = wx.BoxSizer(wx.HORIZONTAL)

```

The two buttons will be added into this horizontal box sizer.

```

okButton = wx.Button(pnl, label="OK")
cancelButton = wx.Button(pnl, label="Cancel")

```

We create two instances of a `wx.Button` widget.

```

hbox.Add(okButton, flag=wx.RIGHT | wx.LEFT,
         border=5)
hbox.Add(cancelButton, flag=wx.RIGHT,
         border=5)

```

Two buttons are added to the horizontal box sizer. We put some space between the buttons, and the cancel button and the right window border.

```

vbox.InsertStretchSpacer(0)

```

We insert a stretchable space into the vertical box sizer. This space grows when the window is vertically resized. It pushes the horizontal box with the buttons to the bottom of the window.

```

vbox.Add(hbox, flag=wx.ALIGN_RIGHT|wx.BOTTOM,
         border=5)

```

The horizontal box sizer is added to the base vertical box sizer. We align the buttons to the right of the box sizer and put some space between the box sizer and the bottom of the window.

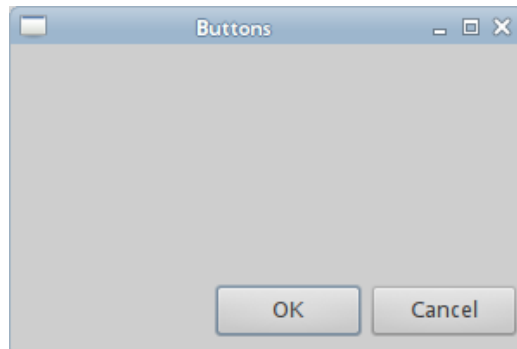


Figure 1.8: Buttons example

Figure 1.8 shows two buttons which reside in the bottom right corner of the window.

1.2.8 New folder example

The next practical example creates a layout that we call a New folder.

Listing 1.9: New folder example

```
def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    hbox1 = wx.BoxSizer(wx.HORIZONTAL)
    pnl.SetSizer(vbox)

    hbox1.Add(wx.StaticText(pnl, label="Name:"),
               flag=wx.RIGHT, border=5)
    hbox1.Add(wx.TextCtrl(pnl), proportion=1)
    vbox.Add(hbox1, flag=wx.EXPAND|wx.ALL,
              border=8)

    hbox2 = wx.BoxSizer(wx.HORIZONTAL)
    hbox2.Add(wx.TextCtrl(pnl, style=wx.TE_MULTILINE),
               proportion=1, flag=wx.EXPAND|wx.ALL, border=8)
    vbox.Add(hbox2, proportion=1, flag=wx.EXPAND)

    hbox3 = wx.BoxSizer(wx.HORIZONTAL)
    hbox3.Add(wx.Button(pnl, label="OK"),
               flag=wx.RIGHT, border=5)
    hbox3.Add(wx.Button(pnl, label="Cancel"))
    vbox.Add(hbox3,
              flag=wx.ALIGN_RIGHT|wx.RIGHT|wx.BOTTOM,
              border=8)
```

The layout is created with one vertical box sizer and three horizontal box sizers. Vertically, the layout is divided into three parts.

```
hbox1.Add(wx.StaticText(pnl, label="Name:"),
           flag=wx.RIGHT, border=5)
hbox1.Add(wx.TextCtrl(pnl), proportion=1)
```

```
vbox.Add(hbox1, flag=wx.EXPAND|wx.ALL,
        border=8)
```

In the first part, we put a static text and a text control into the horizontal box sizer. The text control grows horizontally. There is some space around all sides of the horizontal box.

```
hbox2 = wx.BoxSizer(wx.HORIZONTAL)
hbox2.Add(wx.TextCtrl(pnl, style=wx.TE_MULTILINE),
          proportion=1, flag=wx.EXPAND|wx.ALL, border=8)
vbox.Add(hbox2, proportion=1, flag=wx.EXPAND)
```

In the second part, a multiline text control is added to the second horizontal box sizer. The two `proportion` parameters and their respective `wx.EXPAND` flags make the text control grow in both directions.

```
hbox3 = wx.BoxSizer(wx.HORIZONTAL)
hbox3.Add(wx.Button(pnl, label="OK"),
          flag=wx.RIGHT, border=5)
hbox3.Add(wx.Button(pnl, label="Cancel"))
vbox.Add(hbox3,
          flag=wx.ALIGN_RIGHT|wx.RIGHT|wx.BOTTOM,
          border=8)
```

In the third part, two buttons are placed in the third horizontal box sizer. The buttons are right aligned. There is some space between the buttons, and the buttons and the right and bottom part of the window.

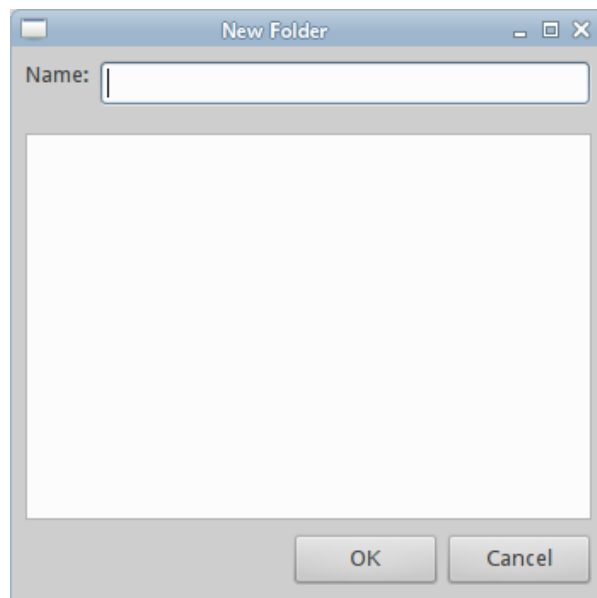


Figure 1.9: New folder example

Figure 1.9 shows the New folder example created with box sizers.

1.2.9 Windows example

The next code example creates a Windows layout that can be found in a real world application.

Listing 1.10: Windows example

```
def InitUI(self):

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    hbox1 = wx.BoxSizer(wx.HORIZONTAL)
    pnl.SetSizer(vbox)

    hbox1.Add(wx.StaticText(pnl, label="Windows"),
               flag=wx.TOP|wx.LEFT, border=5)

    vbox.Add(hbox1)

    hbox2 = wx.BoxSizer(wx.HORIZONTAL)
    hbox2.Add(wx.TextCtrl(pnl, style=wx.TE_MULTILINE),
               proportion=1, flag=wx.EXPAND|wx.ALL, border=5)
    vbox2 = wx.BoxSizer(wx.VERTICAL)
    hbox2.Add(vbox2, flag=wx.TOP|wx.RIGHT, border=5)
    vbox2.Add(wx.Button(pnl, label="OK"),
               flag=wx.BOTTOM, border=3)
    vbox2.Add(wx.Button(pnl, label="Cancel"))

    vbox.Add(hbox2, proportion=1, flag=wx.EXPAND)

    hbox3 = wx.BoxSizer(wx.HORIZONTAL)
    hbox3.Add(wx.Button(pnl, label="Help"))
    hbox3.InsertStretchSpacer(1)
    hbox3.Add(wx.Button(pnl, label="OK"))
    vbox.Add(hbox3,
               flag=wx.EXPAND|wx.LEFT|wx.BOTTOM|wx.RIGHT,
               border=5)
```

The Windows example is created using vertical and horizontal box sizers. The layout is divided into three parts.

```
pnl = wx.Panel(self)

vbox = wx.BoxSizer(wx.VERTICAL)
hbox1 = wx.BoxSizer(wx.HORIZONTAL)
pnl.SetSizer(vbox)
```

The vertical box sizer is the base sizer for the panel widget.

```
hbox1.Add(wx.StaticText(pnl, label="Windows"),
           flag=wx.TOP|wx.LEFT, border=5)
```

In the first part, a static text is added to the first horizontal box sizer. We put some space above the widget and to the left of the widget.

```
hbox2 = wx.BoxSizer(wx.HORIZONTAL)
hbox2.Add(wx.TextCtrl(pnl, style=wx.TE_MULTILINE),
           proportion=1, flag=wx.EXPAND|wx.ALL, border=5)
```

```

vbox2 = wx.BoxSizer(wx.VERTICAL)
hbox2.Add(vbox2, flag=wx.TOP|wx.RIGHT, border=5)
vbox2.Add(wx.Button(pnl, label="OK"),
           flag=wx.BOTTOM, border=3)
vbox2.Add(wx.Button(pnl, label="Cancel"))

```

In the second part, we have a multiline text control and two buttons. We use two additional box sizers: one horizontal and one vertical. We put the text control to the horizontal box sizer with some space around all its sides. Due to the `wx.EXPAND` flag, it takes all the remaining horizontal space.

The buttons are placed in a vertical box sizer and this sizer is added to the horizontal box sizer. We put some space above and to the right of the vertical box sizer.

```

vbox.Add(hbox2, proportion=1, flag=wx.EXPAND)

```

The horizontal box sizer containing the text control and the two buttons is added to the base vertical box sizer. The `wx.EXPAND` flag makes the text control grow vertically, too. The buttons are not affected since their proportion is 0.

```

hbox3 = wx.BoxSizer(wx.HORIZONTAL)
hbox3.Add(wx.Button(pnl, label="Help"))
hbox3.InsertStretchSpacer(1)
hbox3.Add(wx.Button(pnl, label="OK"))
vbox.Add(hbox3,
          flag=wx.EXPAND|wx.LEFT|wx.BOTTOM|wx.RIGHT,
          border=5)

```

The final part has two buttons. The Help button is aligned to the left. The OK button is aligned to the right of the window. This is achieved by putting a stretchable space between the buttons into the horizontal box sizer. We put some space to three sides of the horizontal box containing our two buttons.

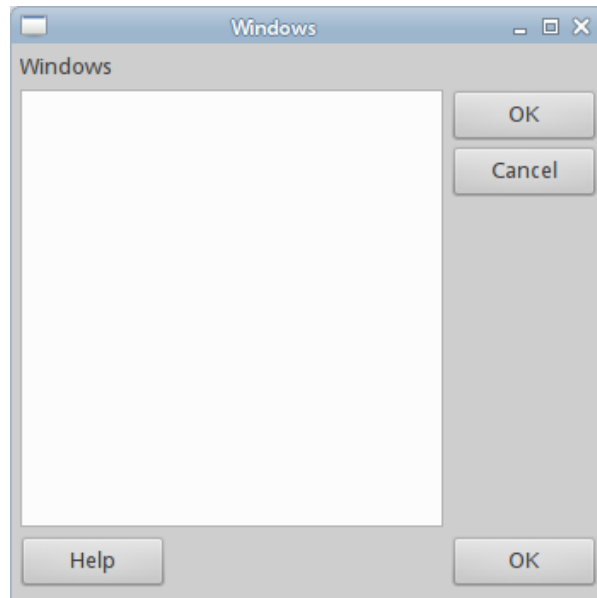


Figure 1.10: Windows example

Figure 1.10 shows the Windows example created with horizontal and vertical box sizers.

1.3 wx.GridBagSizer

A `wx.GridBagSizer` is the most complex and most flexible layout manager in wxPython. The `wx.GridBagSizer` manager places widgets in a grid of rows and columns. Intersections of rows and columns are called cells. Each widget can occupy one or more cells in the grid. The constructor takes two optional parameters. The `vgap` specifies the gap (space) between the rows in the layout manager. The `hgap` specifies the gap between the columns in the layout. Their default value is 0, for no space.

The widgets are added to the layout manager with the `Add` method. The `pos` parameter of the method takes a tuple of two numbers. They provide the row and the column number of the cell where we position the widget. The numbers start from 0.

The `span` parameter can make a widget span multiple rows and columns. Note that there is a difference between assigning a space to a widget and actually occupying it. If we want a widget to stretch over the whole assigned area, we specify the `wx.EXPAND` flag. Moreover, the `flag` parameter can align widgets or add some space around the sides of a widget.

The `border` parameter specifies how much space is added around a widget. It works in combination with the `flag` parameter.

1.3.1 Simple example

First, we have a simple example with the `wx.GridBagSizer` manager.

Listing 1.11: Simple wx.GridBagSizer example

```
def InitUI(self):  
  
    pnl = wx.Panel(self)  
  
    grid = wx.GridBagSizer()  
  
    grid.Add(wx.Button(pnl, label="1:1"), (0, 0))  
    grid.Add(wx.Button(pnl, label="1:2"), (0, 1))  
    grid.Add(wx.Button(pnl, label="1:3"), (0, 2))  
    grid.Add(wx.Button(pnl, label="2:1"), (1, 0))  
    grid.Add(wx.Button(pnl, label="2:2"), (1, 1))  
    grid.Add(wx.Button(pnl, label="2:3"), (1, 2))  
    grid.Add(wx.Button(pnl, label="3:1"), (2, 0))  
    grid.Add(wx.Button(pnl, label="3:2"), (2, 1))  
    grid.Add(wx.Button(pnl, label="3:3"), (2, 2))  
  
    pnl.SetSizer(grid)
```

In this code example, we place nine buttons into the `wx.GridBagSizer` manager.

```
grid = wx.GridBagSizer()
```

The grid bag sizer is created.

```
grid.Add(wx.Button(pnl, label="1:1"), (0, 0))
```

A button is added to the grid bag sizer. The second parameter of the method is the position of the widget within the sizer. It is a tuple containing two values, the row and column number. This button is placed into the first row and the first column. The indexes start from 0.

```
grid.Add(wx.Button(pnl, label="2:3"), (1, 2))
```

This button is placed into the second row, third column.



Figure 1.11: Simple example of wx.GridBagSizer

Figure 1.11 shows a simple `wx.GridBagSizer` example. It shows a group of nine buttons on the window.

1.3.2 Gaps

In the second example, we show how to add gaps between the widgets in the `wx.GridBagSizer`.

Listing 1.12: Gaps in wx.GridBagSizer

```
def InitUI(self):  
  
    pnl = wx.Panel(self)  
  
    grid = wx.GridBagSizer(5, 5)  
  
    grid.Add(wx.Button(pnl, label="Button"), (0, 0))  
    grid.Add(wx.Button(pnl, label="Button"), (0, 1))  
    grid.Add(wx.Button(pnl, label="Button"), (0, 2))  
    grid.Add(wx.Button(pnl, label="Button"), (1, 0))  
    grid.Add(wx.Button(pnl, label="Button"), (1, 1))  
    grid.Add(wx.Button(pnl, label="Button"), (1, 2))  
  
    pnl.SetSizer(grid)
```

Six buttons are placed on the window. There is some space between the buttons.

```
grid = wx.GridBagSizer(5, 5)
```

The grid bag sizer is created. The first parameter is the **vgap** which is a vertical space between the rows of the sizer. The second parameter is the **hgap** which is a horizontal space between the columns of the sizer.



Figure 1.12: Gaps in wx.GridBagSizer

Figure 1.12 shows six buttons on the window with spaces between them.

1.3.3 Spanning

Widgets can span multiple cells in a **wx.GridBagSizer**. The **span** parameter of the **Add** method controls the row and the column span of a widget. With the **span** parameter, the sizer assigns cells to the widget. We also specify the **wx.EXPAND** flag for the widget to actually occupy this space.

Listing 1.13: Spanning multiple columns

```
def InitUI(self):  
  
    pnl = wx.Panel(self)  
  
    grid = wx.GridBagSizer(3, 3)  
  
    grid.Add(wx.Button(pnl, label=""), (0, 0))
```

```

grid.Add(wx.Button(pnl, label=""), (0, 1))
grid.Add(wx.Button(pnl, label=""), (0, 2))
grid.Add(wx.Button(pnl, label=""), (0, 3))

grid.Add(wx.Button(pnl, label=""), (1, 0),
          span=(1, 2), flag=wx.EXPAND)
grid.Add(wx.Button(pnl, label=""), (2, 0),
          span=(1, 3), flag=wx.EXPAND)
grid.Add(wx.Button(pnl, label=""), (3, 0),
          span=(1, 4), flag=wx.EXPAND)

pnl.SetSizer(grid)

```

In the example, we have three buttons that span 2, 3 and 4 columns.

```

grid.Add(wx.Button(pnl, label=""), (0, 0))
grid.Add(wx.Button(pnl, label=""), (0, 1))
grid.Add(wx.Button(pnl, label=""), (0, 2))
grid.Add(wx.Button(pnl, label=""), (0, 3))

```

We add 4 non-spanning buttons to the first row of the `wx.GridBagSizer`. Their purpose is to show us how many columns the other buttons span.

```

grid.Add(wx.Button(pnl, label=""), (1, 0),
          span=(1, 2), flag=wx.EXPAND)

```

We put a button into the second row, first column. The `span` parameter takes a tuple of two numbers. The first is the number of rows to span, the second the number of columns to span. This button spans two columns.

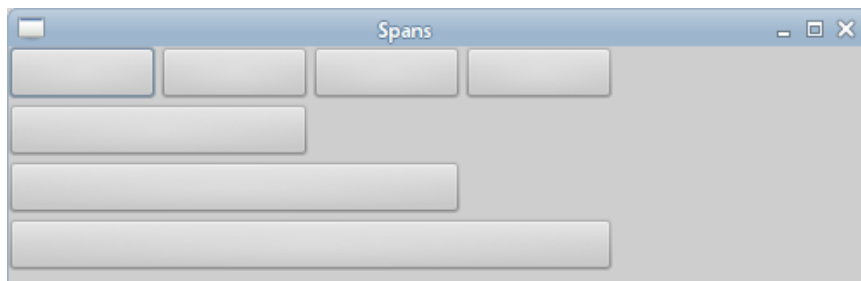


Figure 1.13: Spanning multiple columns

In Figure 1.13 we see buttons that span multiple columns.

1.3.4 Spanning 2

In the second example related to spanning, we have buttons that span both columns and rows.

Listing 1.14: Spanning multiple rows and columns

```

def InitUI(self):

    pnl = wx.Panel(self)

```

```

grid = wx.GridBagSizer(3, 3)

grid.Add(wx.Button(pnl, label=""), (0, 0),
         span=(2, 3), flag=wx.EXPAND)

grid.Add(wx.Button(pnl, label=""), (0, 3))
grid.Add(wx.Button(pnl, label=""), (1, 3))

grid.Add(wx.Button(pnl, label=""), (2, 0),
         span=(1, 3), flag=wx.EXPAND)
grid.Add(wx.Button(pnl, label=""), (2, 3),
         span=(2, 1), flag=wx.EXPAND)

grid.Add(wx.Button(pnl, label=""), (3, 0))
grid.Add(wx.Button(pnl, label=""), (3, 1))
grid.Add(wx.Button(pnl, label=""), (3, 2))

pnl.SetSizer(grid)

```

There are eight buttons in the layout. One of them spans two rows and three columns. The other spans three columns. The final one spans two rows and two columns.

```

grid.Add(wx.Button(pnl, label=""), (2, 0),
         span=(1, 3), flag=wx.EXPAND)

```

This button spans three columns.

```

grid.Add(wx.Button(pnl, label=""), (0, 0),
         span=(2, 3), flag=wx.EXPAND)

```

This button spans two rows and three columns.

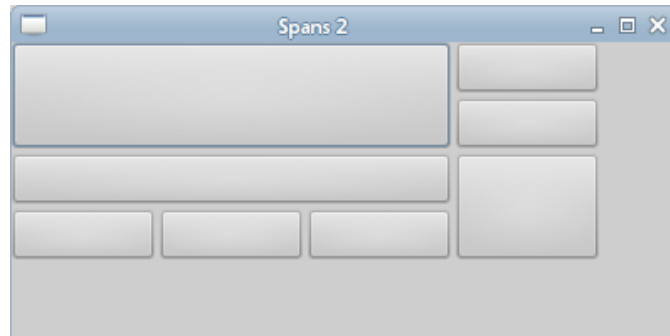


Figure 1.14: Spanning multiple rows and columns

In Figure 1.14 we see buttons that span multiple rows and columns.

1.3.5 Alignments

In this section, we show how to align widgets in the `wx.GridBagSizer`. Widgets are aligned by placing them to the edge cells of the sizer.

Listing 1.15: Alignments in `wx.GridBagSizer`

```
def InitUI(self):

    pnl = wx.Panel(self)

    grid = wx.GridBagSizer(3, 3)

    grid.Add(wx.Button(pnl, label="Left"), (0, 0))
    grid.Add(wx.Button(pnl, label="Center"), (1, 10))
    grid.Add(wx.Button(pnl, label="Right"), (2, 20))
    grid.Add(wx.Button(pnl, label="Stretch"), (3, 0),
              flag=wx.EXPAND, span=(1, 21))

    pnl.SetSizer(grid)
```

There are four buttons in the layout. One is aligned to the left, one to the center, and one to the right. The last button is stretched from the first cell to the last cell of the sizer.

```
grid.Add(wx.Button(pnl, label="Left"), (0, 0))
```

By placing the button in the first row, it is left aligned.

```
grid.Add(wx.Button(pnl, label="Right"), (2, 20))
```

A button placed in the rightmost cell is right aligned in the layout.

```
grid.Add(wx.Button(pnl, label="Stretch"), (3, 0),
          flag=wx.EXPAND, span=(1, 21))
```

This button is stretched over 20 cells. Note the `wx.EXPAND` flag.

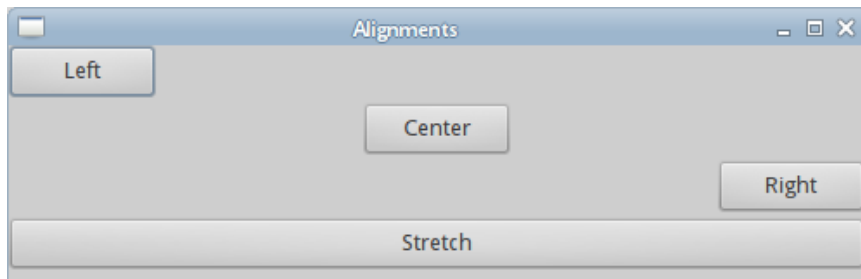


Figure 1.15: Alignments in a `wx.GridBagSizer`

Figure 1.15 shows aligned buttons in a `wx.GridBagSizer`.

1.3.6 Alignments 2

In the following example, we show another way to align widgets in a `wx.GridBagSizer`.

Listing 1.16: Alignments 2 in `wx.GridBagSizer`

```
def InitUI(self):

    pnl = wx.Panel(self)
```

```

grid = wx.GridBagSizer(3, 3)

grid.Add(wx.Button(pnl, label=""), (0, 0),
         span=(2, 3), flag=wx.ALIGN_CENTER)

grid.Add(wx.Button(pnl, label=""), (0, 3))
grid.Add(wx.Button(pnl, label=""), (1, 3))

grid.Add(wx.Button(pnl, label=""), (2, 0))
grid.Add(wx.Button(pnl, label=""), (2, 1))
grid.Add(wx.Button(pnl, label=""), (2, 2))
grid.Add(wx.Button(pnl, label=""), (2, 3))

pnl.SetSizer(grid)

```

One of the buttons is centered in a cell which spans multiple cells of the `wx.GridBagSizer`.

```

grid.Add(wx.Button(pnl, label=""), (0, 0),
         span=(2, 3), flag=wx.ALIGN_CENTER)

```

A button spans 2 rows and 3 columns. We center the button with the `wx.ALIGN_CENTER` flag.

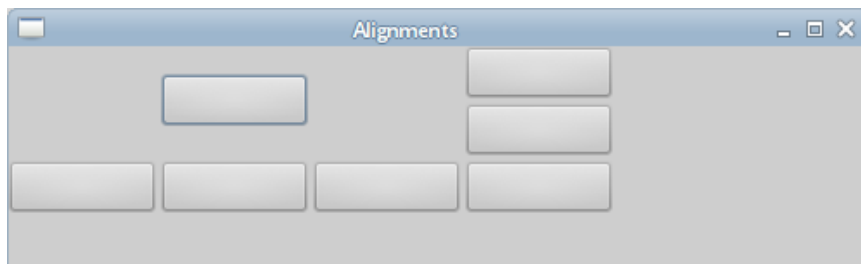


Figure 1.16: A button aligned within a cell

Figure 1.16 shows a button centered in a single cell of the `wx.GridBagSizer`.

1.3.7 Growable rows and columns

The `wx.GridBagSizer` has two methods which take additional space available to the sizer. The `AddGrowableView` adds all extra vertical space to the given row. The `AddGrowableView` adds all extra horizontal space to the given column.

Listing 1.17: Growable rows and columns

```

def InitUI(self):

    pnl = wx.Panel(self)

    grid = wx.GridBagSizer()

    grid.Add(wx.Button(pnl, label="Button"), (0, 0))
    grid.Add(wx.Button(pnl, label="Button"), (0, 1))
    grid.Add(wx.Button(pnl, label="Button"), (0, 2))

```

```

grid.Add(wx.Button(pnl, label="Button"), (0, 3))

grid.Add(wx.TextCtrl(pnl, style=wx.TE_MULTILINE), (1, 0),
          span=(4, 4), flag=wx.EXPAND)

grid.AddGrowableView(2)
grid.AddGrowableView(2)

pnl.SetSizer(grid)

```

We place four buttons into the first row. Below that, we add a multiline text control.

```

grid.Add(wx.Button(pnl, label="Button"), (0, 0))
grid.Add(wx.Button(pnl, label="Button"), (0, 1))
grid.Add(wx.Button(pnl, label="Button"), (0, 2))
grid.Add(wx.Button(pnl, label="Button"), (0, 3))

```

Four buttons are placed into the first row of the sizer.

```

grid.Add(wx.TextCtrl(pnl, style=wx.TE_MULTILINE), (1, 0),
          span=(4, 4), flag=wx.EXPAND)

```

A multiline text control is added below the buttons. It spans four columns and four rows.

```

grid.AddGrowableView(2)
grid.AddGrowableView(2)

```

We make the third row and the third column growable. All additional space is taken by widgets in this particular row and column. This makes the multiline text control stretch over the most area of the layout. The third button also resides in the third column. It is left aligned and additional space is added to the right of the widget. If we specified the `wx.EXPAND` flag for the button, it would stretch horizontally.

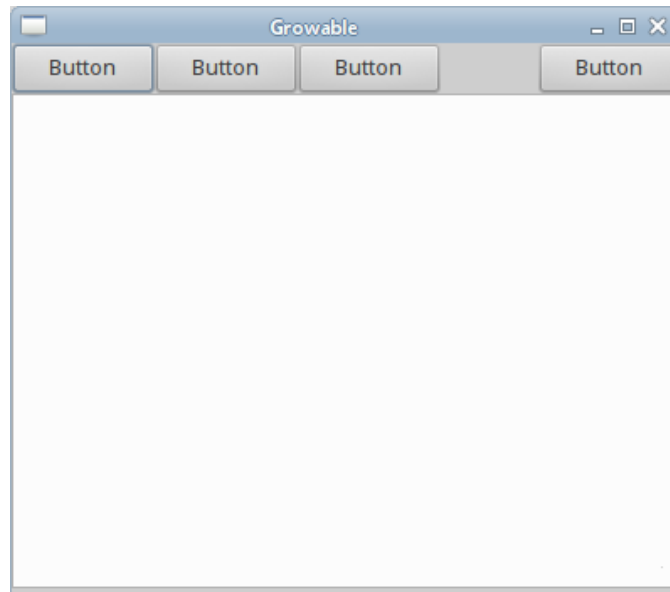


Figure 1.17: Growable rows and columns

Figure 1.17 shows a layout with a growable third row and third column.

1.3.8 New folder example

We have a new folder example created using the `wx.GridBagSizer` layout manager.

Listing 1.18: New folder example

```
def InitUI(self):

    pnl = wx.Panel(self)

    grid = wx.GridBagSizer(8, 8)

    st = wx.StaticText(pnl, label="Name:")
    grid.Add(st, pos=(0, 0),
             flag=wx.LEFT|wx.TOP|wx.ALIGN_CENTER_VERTICAL,
             border=5)
    grid.Add(wx.TextCtrl(pnl), pos=(0, 1), span=(1, 3),
             flag=wx.EXPAND|wx.TOP|wx.RIGHT, border=5)

    tc = wx.TextCtrl(pnl, size=(200, 200),
                     style=wx.TE_MULTILINE)
    grid.Add(tc, pos=(1, 0), span=(2, 4),
             flag=wx.EXPAND|wx.LEFT|wx.RIGHT, border=5)

    grid.Add(wx.Button(pnl, label="OK"), pos=(3, 2),
             flag=wx.BOTTOM, border=5)
    grid.Add(wx.Button(pnl, label="Cancel"), pos=(3, 3),
             flag=wx.BOTTOM|wx.RIGHT, border=5)

    grid.AddGrowableCol(1)
    grid.AddGrowableRow(1)
```

```
pnl.SetSizer(grid)
```

We have a static text, two text controls, and two buttons in the layout.

```
grid = wx.GridBagSizer(8, 8)
```

A `wx.GridBagSizer` manager is created. Eight units of space are distributed between columns and rows of the grid.

```
st = wx.StaticText(pnl, label="Name:")
grid.Add(st, pos=(0, 0),
         flag=wx.LEFT|wx.TOP|wx.ALIGN_CENTER_VERTICAL,
         border=5)
```

A static text is added to the top-left cell of the grid. It is vertically centered, and we put some space to the top and to the left of the widget.

```
grid.Add(wx.TextCtrl(pnl), pos=(0, 1), span=(1, 3),
         flag=wx.EXPAND|wx.TOP|wx.RIGHT, border=5)
```

A text control is placed next to the static text, into the second column of the same row. It spans three columns. A space is added to the top and to the right of the widget.

```
tc = wx.TextCtrl(pnl, size=(200, 200),
                 style=wx.TE_MULTILINE)
grid.Add(tc, pos=(1, 0), span=(2, 4),
         flag=wx.EXPAND|wx.LEFT|wx.RIGHT, border=5)
```

A multiline text control is added to the second row. It spans two rows and four columns. A space is added to the left and to the right of the widget.

```
grid.Add(wx.Button(pnl, label="OK"), pos=(3, 2),
         flag=wx.BOTTOM, border=5)
grid.Add(wx.Button(pnl, label="Cancel"), pos=(3, 3),
         flag=wx.BOTTOM|wx.RIGHT, border=5)
```

Two buttons are placed in the fourth row, below the multiline text control. We put space between the buttons, and the buttons and the bottom of the window.

```
grid.AddGrowbleCol(1)
grid.AddGrowbleRow(1)
```

We make the second row and the second column growable. When the window is resized, the additional space is taken by the two text controls. Without these two lines, there would be a layout of a certain size and all additional space would be distributed around it. Try to comment out the lines and see what happens.

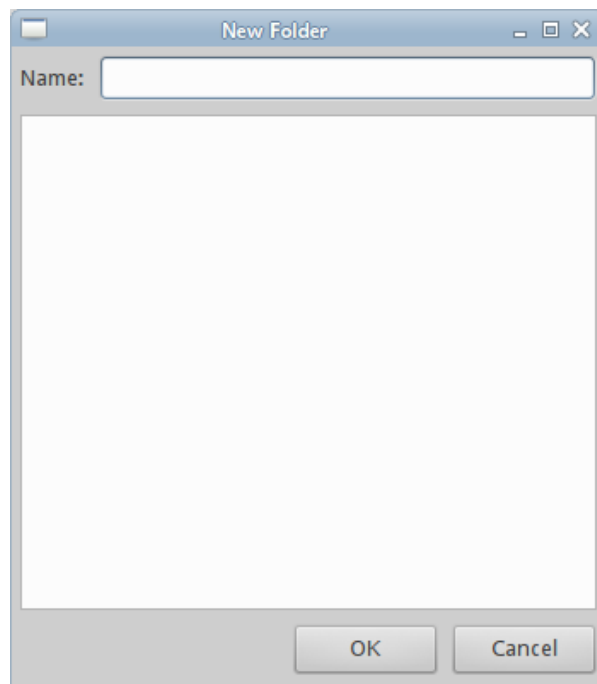


Figure 1.18: New folder example

Figure 1.18 shows the New folder example created with the `wx.GridBagSizer` manager.

1.3.9 Windows example

The second practical example created with the `wx.GridBagSizer` layout manager is the Windows example.

Listing 1.19: Windows example

```
def InitUI(self):

    pnl = wx.Panel(self)

    grid = wx.GridBagSizer(8, 8)

    grid.Add(wx.StaticText(pnl, label="Windows"),
              pos=(0, 0), flag=wx.LEFT|wx.TOP, border=5)

    tc = wx.TextCtrl(pnl, style=wx.TE_MULTILINE)
    grid.Add(tc, pos=(1, 0), span=(3, 2),
              flag=wx.EXPAND|wx.LEFT, border=5)

    grid.Add(wx.Button(pnl, label="Activate"),
              pos=(1, 2), flag=wx.RIGHT, border=5)
    grid.Add(wx.Button(pnl, label="Close"), pos=(2, 2),
              flag=wx.RIGHT, border=5)

    grid.Add(wx.Button(pnl, label="Help"), pos=(4, 0),
              flag=wx.LEFT|wx.BOTTOM, border=5)
```

```

grid.Add(wx.Button(pnl, label="OK"), pos=(4, 2),
         flag=wx.RIGHT|wx.BOTTOM, border=5)

grid.AddGrowbleCol(1)
grid.AddGrowbleRow(2)

pnl.SetSizer(grid)

```

The layout has one static text, one multiline text control, and four buttons.

```
grid = wx.GridBagSizer(8, 8)
```

A `wx.GridBagSizer` with horizontal and vertical gaps between the columns and the rows is created.

```

grid.Add(wx.StaticText(pnl, label="Windows"),
         pos=(0, 0), flag=wx.LEFT|wx.TOP, border=5)

```

A static text widget is added to the top-left cell of the grid. We put five units of space to the left and to the top of the widget.

```

tc = wx.TextCtrl(pnl, style=wx.TE_MULTILINE)
grid.Add(tc, pos=(1, 0), span=(3, 2),
         flag=wx.EXPAND|wx.LEFT, border=5)

```

A multiline text control is added to the first column of the second row. It spans three rows and two columns. We put some space to the left of the widget.

```

grid.Add(wx.Button(pnl, label="Activate"),
         pos=(1, 2), flag=wx.RIGHT, border=5)
grid.Add(wx.Button(pnl, label="Close"), pos=(2, 2),
         flag=wx.RIGHT, border=5)

```

Two buttons are added to the cells next to the multiline text control. Five units of space are added to the right of the buttons.

```

grid.Add(wx.Button(pnl, label="Help"), pos=(4, 0),
         flag=wx.LEFT|wx.BOTTOM, border=5)
grid.Add(wx.Button(pnl, label="OK"), pos=(4, 2),
         flag=wx.RIGHT|wx.BOTTOM, border=5)

```

Two buttons are added to the grid, below the multiline text control. The Help button goes to the leftmost cell, the OK button to the rightmost cell. We put some space between the buttons, and between the buttons and the borders of the window.

```

grid.AddGrowbleCol(1)
grid.AddGrowbleRow(2)

```

We finish the layout by making the second column and the third row growable. All space is taken by the multiline text control. The Close button resides also in the growable row. However, since the `wx.EXPAND` flag for this button is not specified, it does not grow vertically. All space goes below the widget. The Close button is aligned to the top of the cell.

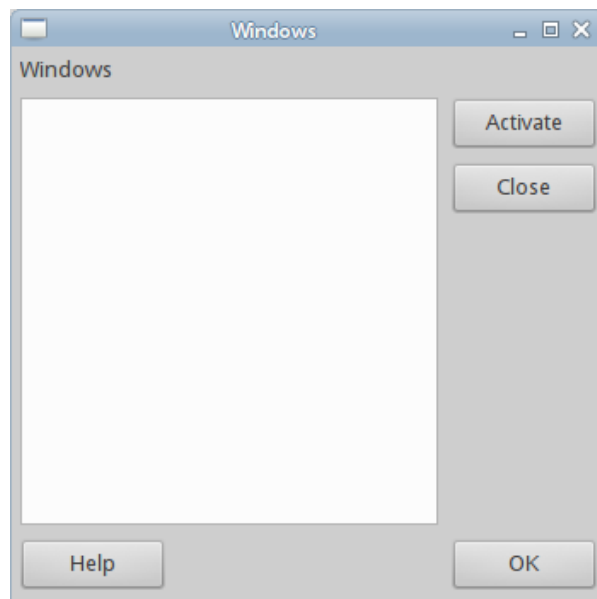


Figure 1.19: Windows example

Figure 1.19 shows the Windows example created with the `wx.GridBagSizer` manager.

Chapter 2

Images

In this chapter, we work with images. We show how to blur, crop and scale an image. We take a screen capture, embed an image in a source file, and create a watermark on an image. We also create a grayscale image and present the art provider.

There are two basic classes for working with images. The `wx.Bitmap` is used to create a platform-dependent bitmap. The `wx.Image` encapsulates a platform-independent bitmap. It has a few methods for manipulating images. The `Images!wx.ImageFromBitmap` method creates an image from a bitmap and the `Images!wx.BitmapFromImage` converts an image to a bitmap.

2.1 Blurring image

In the first example, we create a blurred image.

Listing 2.1: Blurred image

```
def InitUI(self):

    pnl = wx.Panel(self)
    bmp = wx.Bitmap("beckov.jpg")

    blurred_bmp = self.BlurImage(bmp)

    wx.StaticBitmap(pnl, bitmap=blurred_bmp, pos=(2, 2))

def BlurImage(self, bmp):

    img = wx.ImageFromBitmap(bmp)
    sing = img.Blur(2)
    bmp = wx.BitmapFromImage(sing)

    return bmp
```

We load an image from the disk and perform a blur operation on it. The modified image is shown in the `wx.StaticBitmap` widget.

```
bmp = wx.Bitmap("beckov.jpg")
```

A `wx.Bitmap` is created. It takes a JPEG file from the current working directory.

```
blurred_bmp = self.BlurImage(bmp)
```

We perform a blur operation on the bitmap. The result is stored in the `blurred_bmp` variable.

```
wx.StaticBitmap(pnl, bitmap=blurred_bmp, pos=(2, 2))
```

The blurred image is shown on the window in the `wx.StaticBitmap` widget.

```
def BlurImage(self, bmp):  
    img = wx.ImageFromBitmap(bmp)  
    simg = img.Blur(2)  
    bmp = wx.BitmapFromImage(simg)  
  
    return bmp
```

We blur the image in the `BlurImage` method. First, we create a `wx.Image` from a bitmap using the `wx.ImageFromBitmap` method. We perform the blur operation on the `wx.Image` with the `Blur` method. It blurs the image in both horizontal and vertical directions by the specified pixel blur radius. Finally, we convert the image back to the bitmap with the `wx.BitmapFromImage` method.

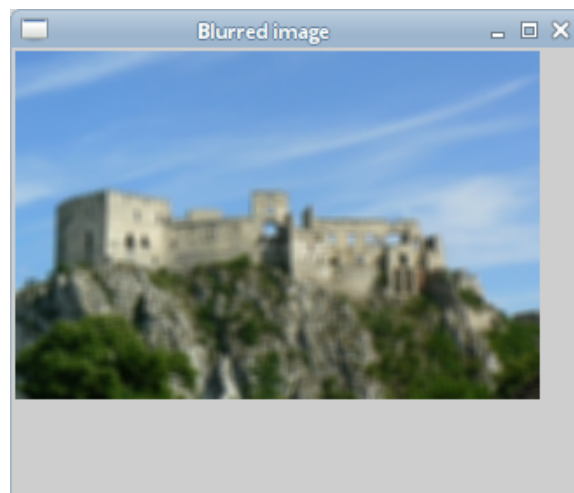


Figure 2.1: Blurred image

Figure 2.1 shows a blurred image.

2.2 Scaling image

In the second example, we scale down an image.

Listing 2.2: Scaled image

```
def InitUI(self):
```

```

        bmp = wx.Bitmap("beckov.jpg")
        wx.StaticBitmap(self, bitmap=bmp, pos=(5, 5))

        scaled_bmp = self.ScaleImage(bmp, 200, 133)

        wx.StaticBitmap(self, bitmap=scaled_bmp,
                        pos=(50, 100))

    def ScaleImage(self, bmp, w, h):

        img = wx.ImageFromBitmap(bmp)
        simg = img.Scale(w, h, wx.IMAGE_QUALITY_HIGH)
        bmp = wx.BitmapFromImage(simg)

        return bmp

```

We show the original image and the scaled image on the window.

```

bmp = wx.Bitmap("beckov.jpg")
wx.StaticBitmap(self, bitmap=bmp, pos=(5, 5))

```

The size of the original image is 300x199 px. We display the image inside the `wx.StaticBitmap` widget.

```

scaled_bmp = self.ScaleImage(bmp, 200, 133)

wx.StaticBitmap(self, bitmap=scaled_bmp,
                pos=(50, 100))

```

In the custom `ScaleImage` method, we scale down the image to 200x133 px. We display it in another `wx.StaticBitmap` over the original one.

```

def ScaleImage(self, bmp, w, h):

    img = wx.ImageFromBitmap(bmp)
    simg = img.Scale(w, h, wx.IMAGE_QUALITY_HIGH)
    bmp = wx.BitmapFromImage(simg)

    return bmp

```

This method performs the scaling operation. We convert a `wx.Bitmap` to the image and call the image's `Scale` method. Then we convert the `wx.Image` back to the `wx.Bitmap`.

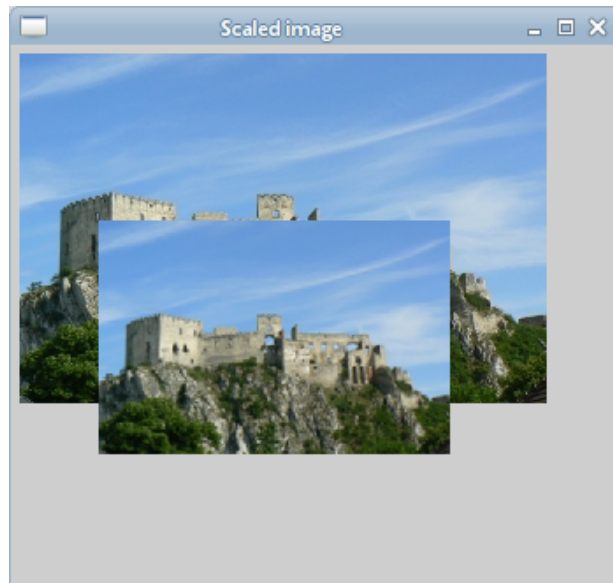


Figure 2.2: Scaled image

Figure 2.2 shows a scaled image over the original image.

2.3 Cropping image

In the third example, we crop an image. A part of the image is cut off and shown next to the original image.

Listing 2.3: Cropped image

```
def InitUI(self):  
    bmp = wx.Bitmap("beckov.jpg")  
    wx.StaticBitmap(self, bitmap=bmp, pos=(5, 5))  
  
    cropped_bmp = self.CropImage(bmp)  
  
    wx.StaticBitmap(self, bitmap=cropped_bmp,  
                    pos=(70, 170))  
  
def CropImage(self, bmp):  
    cbmp = bmp.GetSubBitmap(wx.Rect(10, 50, 250, 100))  
  
    return cbmp
```

This example shows the original image and the image that was cut from it.

```
cropped_bmp = self.CropImage(bmp)
```

We crop a part of the image in the custom `CropImage` method.

```
def CropImage(self, bmp):
    cbmp = bmp.GetSubBitmap(wx.Rect(10, 50, 250, 100))

    return cbmp
```

The `GetSubBitmap` method of the bitmap is used to perform the cropping operation. This time the conversions are not needed.

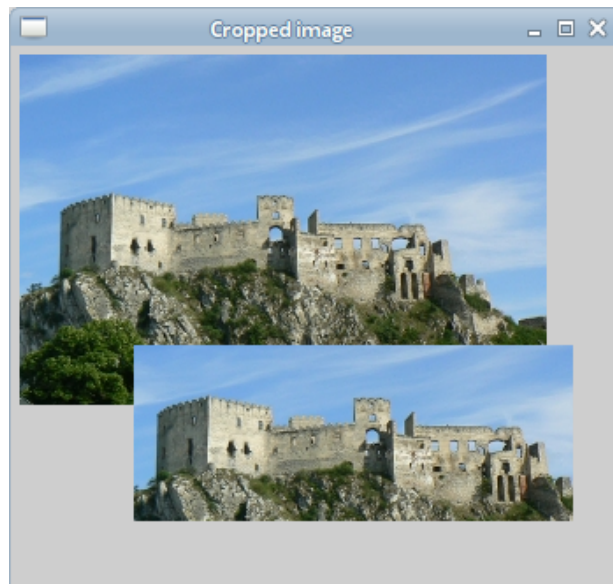


Figure 2.3: Cropped image

Figure 2.3 shows the cropped image next to its original.

2.4 Embedded image

It is possible to embed an image into a source file. The wxPython installation comes with a small program called `img2py.py`. It is available under the `wx/tools` subdirectory of the installation directory. The program converts an image to a Python module.

```
img2py.py smallsid.jpg sidfile.py
```

We convert a JPEG image to a `sidfile.py` module. The module can be loaded into a program at runtime.

Listing 2.4: Embedded image

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

We load an embedded image into the
```



```

program.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sidfile

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.SetTitle("Embedded image")
        self.Centre()

    def InitUI(self):

        pnl = wx.Panel(self)

        sid = sidfile.smallsid.GetBitmap()
        wx.StaticBitmap(pnl, bitmap=sid,
                        pos=(2, 2))

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We display the embedded image in a static bitmap.

```
import sidfile
```

The sidfile.py module is imported.

```
sid = sidfile.smallsid.GetBitmap()
wx.StaticBitmap(pnl, bitmap=sid,
                pos=(2, 2))
```

We get the bitmap from the module using the `GetBitmap` method. The returned bitmap is displayed in the `wx.StaticBitmap` widget.

2.5 Art provider

In our applications we can use either our own resource images or we can use some from the art provider. The wxPython art provider provides some common images for an application. Each bitmap is identified by a unique id. For example,

the `wx.ART_FILE_OPEN` is an id for an image which is used for opening a file.

Listing 2.5: Art provider

```
def InitUI(self):

    tlb = self.CreateToolBar()

    nbmp = wx.ArtProvider.GetBitmap(wx.ART_NEW,
                                    wx.ART_TOOLBAR, (24, 24))
    tlb.AddTool(wx.ID_ANY, "New", nbmp)

    obmp = wx.ArtProvider.GetBitmap(wx.ART_FILE_OPEN,
                                    wx.ART_TOOLBAR, (24, 24))
    tlb.AddTool(wx.ID_ANY, "Open", obmp)

    sbmp = wx.ArtProvider.GetBitmap(wx.ART_FILE_SAVE,
                                    wx.ART_TOOLBAR, (24, 24))
    tlb.AddTool(wx.ID_ANY, "Save", sbmp)

    tlb.Realize()
```

In the example, we show three images from the art provider on the toolbar.

```
tlb = self.CreateToolBar()
```

The images are displayed on the toolbar widget.

```
nbmp = wx.ArtProvider.GetBitmap(wx.ART_NEW,
                                wx.ART_TOOLBAR, (24, 24))
```

We use the `wx.ArtProvider.GetBitmap` method to get the bitmap for the new tool button. The id for this button is `wx.ART_NEW`. The second parameter of the method is the client which displays the image. In our case, it is a toolbar. The third parameter is the size of the image.

```
tlb.AddTool(wx.ID_ANY, "New", nbmp)
```

The bitmap is added to the toolbar using the `AddTool` method.

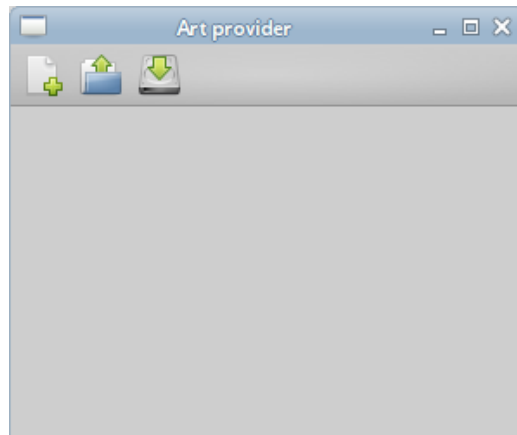


Figure 2.4: Art provider

Figure 2.4 shows three icons on the toolbar from the art provider.

2.6 Grayscale image

In the following example, we create a grayscale image by directly manipulating the image pixels. Note that the `wx.Image` class has a method for converting an image to grayscale. We create a grayscale image for demonstration purposes.

Listing 2.6: Grayscale image

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program, we create a grayscale
image.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.SetTitle("Gray scale")
        self.Centre()

    def InitUI(self):

        self.pnl = wx.Panel(self)
        self.pnl.Bind(wx.EVT_PAINT, self.OnPaint)
```

```

        self.sid = wx.Bitmap("smallsid.jpg")
        self.sidg = self.GrayScale(self.sid)

    def OnPaint(self, e):

        sender = e.GetEventObject()
        dc = wx.PaintDC(sender)

        dc.DrawBitmap(self.sid, 20, 20)
        dc.DrawBitmap(self.sidg, 190, 20)

    def GrayScale(self, img):

        oi = img.ConvertToImage()
        gi = oi.Copy()
        w, h = gi.GetSize()

        for x in range(w):
            for y in range(h):

                r = gi.GetRed(x, y)
                g = gi.GetGreen(x, y)
                b = gi.GetBlue(x, y)

                gr = (r + g + b) // 3

                gi.SetRGB(x, y, gr, gr, gr)

        bmp = wx.Bitmap(gi)

        return bmp

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We draw two images on the window: the original image and its grayscale version.

```

self.sid = wx.Bitmap("smallsid.jpg")
self.sidg = self.GrayScale(self.sid)

```

We create a `wx.Bitmap` from a JPEG file. This bitmap is converted to grayscale inside the custom `GrayScale` method.

```

def OnPaint(self, e):

    sender = e.GetEventObject()
    dc = wx.PaintDC(sender)

```

```
dc.DrawBitmap(self.sid, 20, 20)
dc.DrawBitmap(self.sidg, 190, 20)
```

This time the images are drawn on the window in the `OnPaint` method.

```
oi = wx.ImageFromBitmap(img)
gi = oi.Copy()
```

Inside the `GrayScale` method, we create an image from the bitmap. A copy of the image is returned with the `Copy` method. We do our modifications on the copy of the image.

```
w, h = gi.GetSize()
```

We get the width and height of the image using the `GetSize` method.

```
for x in range(w):
    for y in range(h):

        r = gi.GetRed(x, y)
        g = gi.GetGreen(x, y)
        b = gi.GetBlue(x, y)

        gr = (r + g + b) / 3

        gi.SetRGB(x, y, gr, gr, gr)
```

We go through the pixels of the image and modify them by a specific formula. There are several algorithms for creating grayscale images; we use the simplest one. The formula creates a grayscale image by averaging the red, green, and blue parts of each pixel in the image.

```
bmp = wx.BitmapFromImage(gi)
```

We convert the image back to a bitmap.



Figure 2.5: Grayscale image

Figure 2.5 shows an original image and a grayscale version of the same image.

2.7 Watermark

The text written on an image is called a watermark. Watermarks are used to identify images. They could be copyright notices or image creation times.

Listing 2.7: Watermark

```
def InitUI(self):

    self.pnl = wx.Panel(self)

    self.img = wx.Bitmap("beckov.jpg", wx.BITMAP_TYPE_JPEG)
    wi = self.CreateWatermark(self.img)

    wx.StaticBitmap(self.pnl, bitmap=wi, pos=(2, 2))

def CreateWatermark(self, bmp):

    imgs = wx.lib.wxcairo.ImageSurfaceFromBitmap(bmp)
    cr = cairo.Context(imgs)
    cr.set_font_size(11)
    cr.set_source_rgb(0.9, 0.9, 0.9)
    cr.move_to(20, 30)
    cr.show_text("Beckov 2012, (c) Jan Bodnar")
    cr.stroke()

    bmp = wx.lib.wxcairo.BitmapFromImageSurface(imgs)

    return bmp
```

We draw copyright information on an image.

```
self.img = wx.Bitmap("beckov.jpg", wx.BITMAP_TYPE_JPEG)
wi = self.CreateWatermark(self.img)
```

The `CreateWatermark` method takes a JPEG image and draws a watermark on it. It returns the modified image.

```
wx.StaticBitmap(self.pnl, bitmap=wi, pos=(2, 2))
```

The image with a watermark is displayed in a static bitmap.

```
def CreateWatermark(self, bmp):

    imgs = wx.lib.wxcairo.ImageSurfaceFromBitmap(bmp)
    cr = cairo.Context(imgs)
    ...
```

Inside the `CreateWatermark` method, we create a cairo image surface from the bitmap using the `ImageSurfaceFromBitmap` method. From this surface, we create a cairo drawing context.

```
cr.set_font_size(11)
cr.set_source_rgb(0.9, 0.9, 0.9)
cr.move_to(20, 30)
cr.show_text("Beckov 2012, (c) Jan Bodnar")
cr.stroke()
```

We draw the text on the drawing context. It is a small text in white colour.

```
bmp = wx.lib.wxcairo.BitmapFromImageSurface(imgs)
```

We create a bitmap back from the image surface.



Figure 2.6: Watermark

Figure 2.6 shows a copyright notice on the image.

2.8 Screenshot

In the last example of the Images chapter, we make a small application that takes a screenshot.

Listing 2.8: Screenshot

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

This program captures a screen shot.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class s:

    IMG_WIDTH = 350
    IMG_HEIGHT = 250

class Example(wx.Frame):

    def __init__(self, *args, **kw):
```

```

super(Example, self).__init__(*args, **kw)

self.InitUI()
self.SetSize((390, 350))
self.SetTitle("Screen capture")
self.Centre()

def InitUI(self):

    self.mp = wx.Panel(self)
    self.bmp = wx.Bitmap(s.IMG_WIDTH,
        s.IMG_HEIGHT, 24)
    self.pnl = wx.Panel(self.mp, size=(s.IMG_WIDTH,
        s.IMG_HEIGHT))
    self.pnl.Bind(wx.EVT_PAINT, self.OnPaint)

    self.btn = wx.Button(self.mp,
        label="Take screenshot")
    self.btn.Bind(wx.EVT_BUTTON, self.CaptureScreen)

    vbox = wx.BoxSizer(wx.VERTICAL)
    vbox.Add(self.pnl, 1, wx.ALIGN_CENTER)
    vbox.Add(self.btn, 0, wx.ALIGN_CENTER | wx.TOP |
        wx.BOTTOM, 25)
    self.mp.SetSizer(vbox)

def OnPaint(self, e):

    sender = e.GetEventObject()
    dc = wx.PaintDC(sender)

    dc.DrawBitmap(self.bmp, 5, 5)

def ScaleBitmap(self, bitmap, w, h):

    img = bitmap.ConvertToImage()
    img = img.Scale(w, h, wx.IMAGE_QUALITY_HIGH)
    bmp = wx.Bitmap(img)

    return bmp

def CaptureScreen(self, e):

    sw, sh = wx.DisplaySize()

    sdc = wx.ScreenDC()
    bitmap = wx.Bitmap(sw, sh, 24)
    mdc = wx.MemoryDC()
    mdc.SelectObject(bitmap)
    mdc.Blit(0, 0, sw, sh, sdc, 0, 0)
    mdc.SelectObject(wx.NullBitmap)

    self.bmp = self.ScaleBitmap(bitmap, s.IMG_WIDTH,
        s.IMG_HEIGHT)
    self.pnl.Refresh()

def main():

```



```

app = wx.App()
ex = Example(None)
ex.Show()
app.MainLoop()

if __name__ == "__main__":
    main()

```

We have a button and an image drawn on the window. At the beginning the image is an empty black rectangle. Clicking on the button we capture a screenshot, which is then displayed on the window.

```

class s(object):

    IMG_WIDTH = 350
    IMG_HEIGHT = 250

```

We store two values in the `s` class: the image width and height.

```

self.bmp = wx.EmptyBitmap(s.IMG_WIDTH,
    s.IMG_HEIGHT, 24)

```

An empty bitmap is created. We draw our screenshots on this bitmap. The third parameter is the image depth. The number is platform dependent. Value 24 works both on Linux and Windows.

```

self.btn = wx.Button(self.mp,
    label="Take screenshot")
self.btn.Bind(wx.EVT_BUTTON, self.CaptureScreen)

```

We create a button widget. The `CaptureScreen` method is called when we click on the button.

```

def OnPaint(self, e):

    sender = e.GetEventObject()
    dc = wx.PaintDC(sender)

    dc.DrawBitmap(self.bmp, 5, 5)

```

In the `OnPaint` method, we draw the bitmap on the panel. First, it is an empty bitmap, later, it is the screenshot.

```

def ScaleBitmap(self, bitmap, w, h):

    img = wx.ImageFromBitmap(bitmap)
    img = img.Scale(w, h, wx.IMAGE_QUALITY_HIGH)
    bmp = wx.BitmapFromImage(img)

    return bmp

```

In the `ScaleBitmap` method, we scale the screenshot. The screen is 1280x800, for example. We need to scale the screenshot down to fit the panel on the window.

```

def CaptureScreen(self, e):

```

```

    sw, sh = wx.DisplaySize()
    ...

```

In the `CaptureScreen` method, we take a snapshot of the entire screen. The `wx.DisplaySize` returns the size of our display.

```

sdc = wx.ScreenDC()

```

We create a screen device context. It is used to work with the screen which is a special kind of a GUI object.

```

bitmap = wx.EmptyBitmap(sw, sh, 24)
mdc = wx.MemoryDC()
mdc.SelectObject(bitmap)
mdc.Blit(0, 0, sw, sh, sdc, 0, 0)
mdc.SelectObject(wx.NullBitmap)

```

We create an empty bitmap in the memory. Using the `Blit` method, we copy the entire screen to the empty bitmap.

```

self.bmp = self.ScaleBitmap(bitmap, s.IMG_WIDTH,
                             s.IMG_HEIGHT)
self.pnl.Refresh()

```

We scale the bitmap to fit the panel on the window and refresh the panel. Now the screenshot is visible on the panel.

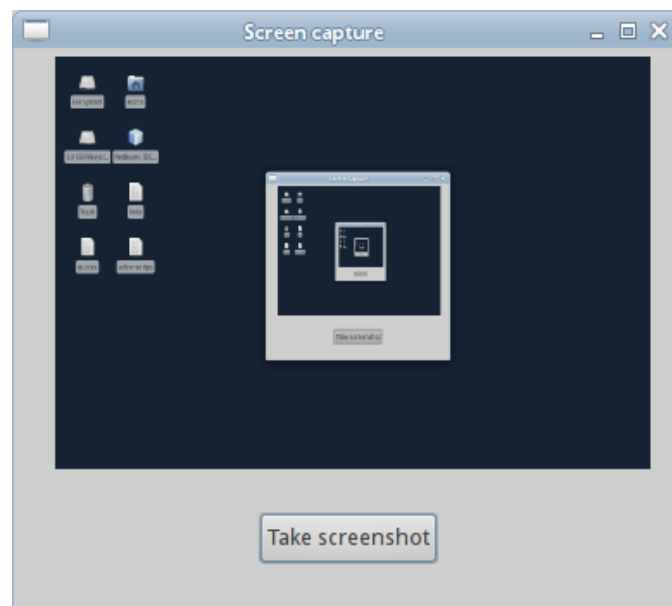


Figure 2.7: Screenshot

Figure 2.7 shows a screenshot displayed on the window.

Chapter 3

Custom widgets

Toolkits usually provide only the most common widgets like buttons, text widgets, sliders etc. No toolkit can provide all possible widgets. Programmers must create other widgets by themselves. There are two possibilities: a) we can modify or enhance an existing widget, or b) we can create a custom widget from scratch.

In this chapter, we are going to create three custom widgets: a ProgressMeter widget, a Thermometer widget, and a ColorWheel widget. These three custom widgets will be created from scratch. We use the Cairo library to draw the widgets.

3.1 ProgressMeter widget

In this section, we create a progress meter. A progress meter is a widget which shows a progress of an event. Our progress meter has a circular shape.

Listing 3.1: ProgressMeter widget

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a custom
progress meter widget.

Author: Jan Bodnar
Website: zetcode.com
"""

import math
import wx
import cairo
import wx.lib.wxcairo

class pm:
    """Stores constants to avoid
    global values."""
```

```

SPEED = 30
INIT_X = 70
INIT_Y = 70

```

```

class ProgressMeter(wx.Panel):

    def __init__(self, parent):
        super(ProgressMeter, self).__init__(parent,
            style=wx.BORDER_SIMPLE)

        self.InitUI()

    def InitUI(self):

        self.SetDoubleBuffered(True)

        self.angle = 0
        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_TIMER, self.OnTimer)

        self.timer = wx.Timer(self, 1)
        self.timer.Start(pm.SPEED)

    def OnTimer(self, e):
        """Controls the progress of the meter."""

        self.angle += 1

        if self.angle >= 360:
            self.timer.Stop()

        self.Refresh()

    def OnPaint(self, e):
        """Reacts to paint events."""

        dc = wx.PaintDC(self)
        cr = wx.lib.wxcairo.ContextFromDC(dc)

        self.DrawBackground(cr)
        self.DrawMeter(cr)
        self.DrawText(cr)

    def DrawBackground(self, cr):
        """Draws the background of the custom widget."""

        w, h = self.GetClientSize()

        cr.translate(pm.INIT_X, pm.INIT_Y)

        radial = cairo.RadialGradient(-30, -30, 10,
            -30, -30, 90)
        radial.add_color_stop_rgba(0, 1, 1, 1, 1)
        radial.add_color_stop_rgba(1, 0.6, 0.6, 0.6, 1)
        cr.set_source(radial)
        cr.arc(0, 0, 40, 0, math.pi * 2)
        cr.fill()

```

```

def DrawMeter(self, cr):
    """Draws the arc in blue colour."""

    cr.set_source_rgb(0.3, 0.5, 1)
    cr.move_to(0, 0)
    cr.line_to(33, 0)
    cr.arc(0, 0, 33, 0, self.angle * math.pi / 180)
    cr.close_path()
    cr.fill()

def DrawText(self, cr):
    """Draws the percentage of the task finished."""

    self.per = int(self.angle / 3.6)
    text = f"{self.per}%"

    (_, _, w, h, _, _) = cr.text_extents(text)

    cr.set_source_rgb(0, 0, 0)
    cr.move_to(-w/2, h/2)
    cr.show_text(text)

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.cw = ProgressMeter(self)
        self.SetSize((360, 270))
        self.SetTitle("ProgressMeter")
        self.Centre()

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

A radial gradient is used for the background of the progress meter. It creates an illusion of a three dimensional object. The progress of an event is illustrated by a pie of a blue colour. The pie grows until it fills the entire inner circle. The progress of an event is also marked by the percentage shown in the middle of the meter.

```

def OnTimer(self, e):

```

```

        """Controls the progress of the meter."""

        self.angle += 1

        if self.angle >= 360:
            self.timer.Stop()

        self.Refresh()

```

This is one cycle of the progress. The `angle` variable is incremented. If the angle equals a full circle, we stop the timer. The `Refresh` method redraws the custom widget to update the meter to the current angle value.

```

def OnPaint(self, e):
    """Reacts to paint events."""

    dc = wx.PaintDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)

    self.DrawBackground(cr)
    self.DrawMeter(cr)
    self.DrawText(cr)

```

The drawing is divided into three steps. The drawing of the background, of the meter and of the text.

```

def DrawBackground(self, cr):
    """Draws the background of the custom widget."""

    w, h = self.GetClientSize()

    cr.translate(pm.INIT_X, pm.INIT_Y)

    radial = cairo.RadialGradient(-30, -30, 10,
                                  -30, -30, 90)
    radial.add_color_stop_rgba(0, 1, 1, 1, 1)
    radial.add_color_stop_rgba(1, 0.6, 0.6, 0.6, 1)
    cr.set_source(radial)
    cr.arc(0, 0, 40, 0, math.pi * 2)
    cr.fill()

```

In the `DrawBackground` method, we draw a circular shape filled with a radial gradient. An illusion of a 3D object is created.

```

def DrawMeter(self, cr):
    """Draws the arc in blue colour."""

    cr.set_source_rgb(0.3, 0.5, 1)
    cr.move_to(0, 0)
    cr.line_to(33, 0)
    cr.arc(0, 0, 33, 0, self.angle * math.pi / 180)
    cr.close_path()
    cr.fill()

```

This method draws the meter. We use a blue fill colour. The end angle is stored in the `angle` variable which is incremented automatically in the `OnTimer` method.

```

def DrawText(self, cr):
    """Draws the percentage of the task finished."""

```

```

self.per = int(self.angle / 3.6)
text = f"{self.per}%"

(_, _, w, h, _, _) = cr.text_extents(text)

cr.set_source_rgb(0, 0, 0)
cr.move_to(-w/2, h/2)
cr.show_text(text)

```

We draw a text on the meter. The text is the percentage of the completed task. We use the `text_extents` method to get the text extents. Only the width and height of the text are needed. They are used to center the text on the progress meter.

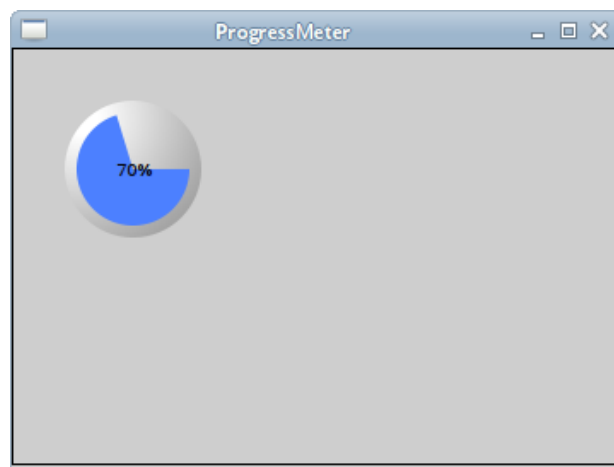


Figure 3.1: ProgressMeter widget

Figure 3.1 shows a custom ProgressMeter widget.

3.2 Thermometer widget

In the following example, we create a Thermometer widget. The widget visually represents temperature values.

Listing 3.2: Thermometer widget

```

#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a custom
thermometer widget.

Author: Jan Bodnar
Website: zetcode.com
"""

import math

```

```

import wx
import cairo
import wx.lib.wxcairo
from colorsys import rgb_to_hsv, hsv_to_rgb

class t:
    """Stores constants to avoid global values."""

    TEMP_OFFSET = 10
    TEMP_X = -4
    TEMP_W = 8
    TEXT_X = 10
    SCALING_FACTOR = 300.0
    SATURATION_DROP = 100/255.0
    SCALES_HEIGHT = 224
    SCALES_START = 252
    SCALES_END = 28
    SCALES_UNIT_HEIGHT = 28
    SCALES_SMALL_UNIT_HEIGHT = 7
    LINE_LENGTH_BIG = 12
    LINE_LENGTH_MEDIUM = 8
    LINE_LENGTH_SMALL = 5

class Thermometer(wx.Panel):

    def __init__(self, parent):
        super(Thermometer, self).__init__(parent,
            style=wx.BORDER_SIMPLE)

        self.InitValues()
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def InitValues(self):
        """Initiates values."""

        self.tmpr = 0
        self.normal = 25
        self.critical = 75.0
        self.m_min = 0
        self.m_max = 80.0

    def OnPaint(self, e):
        """Calls methods to draw the custom widget."""

        dc = wx.PaintDC(self)
        cr = wx.lib.wxcairo.ContextFromDC(dc)

        self.InitDrawing(cr)
        self.DrawTemperature(cr)
        self.DrawBackground(cr)

    def InitDrawing(self, cr):
        """Initiates drawing."""

        self.SetDoubleBuffered(True)

        w, h = self.GetClientSize()

```



```

        cr.set_antialias(cairo.ANTIALIAS_DEFAULT)
        cr.translate(w/2.0, 0)
        cr.scale(h/t.SCALING_FACTOR, h/t.SCALING_FACTOR)

def SetTemperature(self, val):
    """Stores the value from the slider."""

    self.tmpr = val

def DrawBackground(self, cr):
    """Draws the background, the scale of
    the thermometer."""

    cr.set_source_rgba(0, 0, 0, 1)
    cr.set_line_width(1)

    cr.curve_to(-7, 257, -12, 263, -12, 268)
    cr.curve_to(-12, 268, -12, 278, 0, 280)
    cr.curve_to(0, 280, 12, 278, 12, 268)
    cr.curve_to(12, 268, 12, 263, 7, 257)

    cr.line_to(7, 25)
    cr.curve_to(7, 25, 7, 12, 0, 12)
    cr.curve_to(0, 12, -7, 12, -7, 25)
    cr.line_to(-7, 257)

    cr.stroke_preserve()

    lg = cairo.LinearGradient(-85, 0, 80, 0)
    lg.set_extend(cairo.EXTEND_REFLECT)
    lg.add_color_stop_rgba(0.1, 0, 0.58, 1, 0.66)
    lg.add_color_stop_rgba(0.4, 1, 1, 1, 0)
    lg.add_color_stop_rgba(0.7, 0, 0.58, 1, 0.66)
    cr.set_source(lg)
    cr.fill()

    cr.set_source_rgb(0, 0, 0)

    for i in range(33):

        line_len = t.LINE_LENGTH_BIG

        if i % 4:
            line_len = t.LINE_LENGTH_MEDIUM
            cr.set_line_width(0.8)

        if i % 2:
            line_len = t.LINE_LENGTH_SMALL
            cr.set_line_width(0.6)

        cr.move_to(-t.SCALES_SMALL_UNIT_HEIGHT,
            t.SCALES_END + i*t.SCALES_SMALL_UNIT_HEIGHT)
        cr.line_to(-t.SCALES_SMALL_UNIT_HEIGHT+line_len,
            t.SCALES_END + i*t.SCALES_SMALL_UNIT_HEIGHT)
        cr.stroke()

    for i in range(9):

        num = self.m_min + i*(self.m_max-self.m_min)/8.0

```

```

        val = "%d" % num

        _, _, _, font_h, _, _ = cr.text_extents(val)

        cr.move_to(t.TEXT_X, t.SCALES_START -
                    i*t.SCALES_UNIT_HEIGHT + font_h/2)
        cr.show_text(val)

def DrawTemperature(self, cr):
    """Draws the temperature indicator."""

    if self.tmpr >= self.critical:
        col = (1, 0, 0)

    elif self.tmpr >= self.normal:
        col = (0, 0.7, 0)

    else:
        col = (0, 0, 1)

    r, g, b, = col

    bulbGrad = cairo.RadialGradient(-8, 250, 10,
                                     -8, 250, 30)
    bulbGrad.set_extend(cairo.EXTEND_REFLECT)
    bulbGrad.add_color_stop_rgb(1, r, g, b)
    bulbGrad.add_color_stop_rgba(0, r, g, b, 0)

    cr.new_path()
    cr.arc(0, 268, 9, 0, math.radians(360))
    cr.set_source(bulbGrad)
    cr.fill()

    scaleGrad = cairo.LinearGradient(0, 0, 5, 0)
    scaleGrad.set_extend(cairo.EXTEND_REFLECT)

    h, s, v = rgb_to_hsv(r, g, b)

    r2, g2, b2 = hsv_to_rgb(h, s - t.SATURATION_DROP, v)
    scaleGrad.add_color_stop_rgb(1, r, g, b)
    scaleGrad.add_color_stop_rgb(0, r2, g2, b2)

    factor = self.tmpr-self.m_min
    factor = (factor/self.m_max)-self.m_min

    temp = t.SCALES_HEIGHT * factor
    height = temp + t.TEMP_OFFSET

    cr.set_source(scaleGrad)
    cr.rectangle(t.TEMP_X, t.SCALES_START +
                 t.TEMP_OFFSET - height, t.TEMP_W, height)
    cr.fill()

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.SetSize((300, 380))

```

```

        self.SetTitle("Thermometer")
        self.Centre()

def InitUI(self):
    """Builds the user interface consisting of the
       custom widget and the slider."""

    pnl = wx.Panel(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    pnl.SetSizer(vbox)

    self.cw = Thermometer(pnl)
    vbox.Add(self.cw, proportion=1, flag=wx.EXPAND)

    sld = wx.Slider(pnl, value=0, minValue=0,
                    maxValue=80, style=wx.SL_HORIZONTAL)
    vbox.Add(sld, proportion=0,
             flag=wx.EXPAND|wx.TOP|wx.BOTTOM, border=10)

    sld.Bind(wx.EVT_SCROLL, self.OnSliderScroll)

    self.SetMinSize((120, 120))

def OnSliderScroll(self, e):
    """Retrieves the value of the slider and sets it
       to the custom widget variable."""

    obj = e.GetEventObject()
    val = obj.GetValue()

    self.cw.SetTemperature(val)
    self.cw.Refresh()

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

There are two widgets on the window. A custom Thermometer and a slider widget. The slider widget controls the temperature of the Thermometer.

```

class t:
    """Stores constants to avoid global values."""

    TEMP_OFFSET = 10
    TEMP_X = -4
    TEMP_W = 8
    ...

```

The `t` class is used to store constants.

```
def InitValues(self):
    """Initiates values."""

    self.tmpr = 0
    self.normal = 25
    self.critical = 75.0
    self.m_min = 0
    self.m_max = 80.0
```

In the `InitValues` method, we do some initializations. From 0 to `self.normal`, we draw the temperature in blue colour. From `self.normal` to `self.critical` value, we draw in green colour. Temperatures higher than `self.critical` are drawn in red colour. The `self.m_min` and `self.m_max` are the minimal and maximal values on our scale.

```
def OnPaint(self, e):
    """Calls methods to draw the custom widget."""

    dc = wx.PaintDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)

    self.InitDrawing(cr)
    self.DrawTemperature(cr)
    self.DrawBackground(cr)
```

The drawing of the widget is delegated into three methods. The `InitDrawing` does some preparatory work. The `DrawTemperature` draws the temperature of the Thermometer and the `DrawBackground` draws the background of the Thermometer.

```
def InitDrawing(self, cr):
    """Initiates drawing."""

    self.SetDoubleBuffered(True)

    w, h = self.GetClientSize()

    cr.set_antialias(cairo.ANTIALIAS_DEFAULT)
    cr.translate(w/2.0, 0)
    cr.scale(h/t.SCALING_FACTOR, h/t.SCALING_FACTOR)
```

We do some preliminary work before the actual drawing. Antialiasing is used in our drawings. We translate the drawing into the middle of the window. The `scale` method makes the custom widget grow or shrink when the whole window grows or shrinks.

```
def SetTemperature(self, val):
    """Stores the value from the slider."""

    self.tmpr = val
```

The `SetTemperature` is a setter method for the `tmpr` variable. The variable stores the current temperature. The method is called when we select a new value with the slider widget.

```
cr.curve_to(-7, 257, -12, 263, -12, 268)
cr.curve_to(-12, 268, -12, 278, 0, 280)
cr.curve_to(0, 280, 12, 278, 12, 268)
```

```

cr.curve_to(12, 268, 12, 263, 7, 257)

cr.line_to(7, 25)
cr.curve_to(7, 25, 7, 12, 0, 12)
cr.curve_to(0, 12, -7, 12, -7, 25)
cr.line_to(-7, 257)

cr.stroke_preserve()

```

These code lines draw the outline of the scale. The `curve_to` method draws Bézier curves. The `line_to` method draws straight lines. Note that we have moved the drawing origin. Therefore, we work with negative values.

```

lg = cairo.LinearGradient(-85, 0, 80, 0)
lg.set_extend(cairo.EXTEND_REFLECT)
lg.add_color_stop_rgba(0.1, 0, 0.58, 1, 0.66)
lg.add_color_stop_rgba(0.4, 1, 1, 1, 0)
lg.add_color_stop_rgba(0.7, 0, 0.58, 1, 0.66)
cr.set_source(lg)
cr.fill()

```

A linear gradient is used to fill the inside of the scale. The values were chosen by some experimentation.

```

for i in range(33):

    line_len = t.LINE_LENGTH_BIG

    if i % 4:
        line_len = t.LINE_LENGTH_MEDIUM
        cr.set_line_width(0.8)

    if i % 2:
        line_len = t.LINE_LENGTH_SMALL
        cr.set_line_width(0.6)

    cr.move_to(-t.SCALES_SMALL_UNIT_HEIGHT,
               t.SCALES_END + i*t.SCALES_SMALL_UNIT_HEIGHT)
    cr.line_to(-t.SCALES_SMALL_UNIT_HEIGHT+line_len,
               t.SCALES_END + i*t.SCALES_SMALL_UNIT_HEIGHT)
    cr.stroke()

```

In the above for loop, we draw the lines on the scale. There are three kinds of lines; they differ by their length and width.

```

for i in range(9):

    num = self.m_min + i*(self.m_max-self.m_min)/8.0
    val = "%d" % num

    _, _, _, font_h, _, _ = cr.text_extents(val)

    cr.move_to(t.TEXT_X, t.SCALES_START -
               i*t.SCALES_UNIT_HEIGHT + font_h/2)
    cr.show_text(val)

```

Next to the lines, we draw numbers: 0, 10, 20, .., 80. We get the text metrics to get the size of the font. The underscores are used to discard values that we do not need in our computation.

```

if self.tmpr >= self.critical:
    col = (1, 0, 0)

elif self.tmpr >= self.normal:
    col = (0, 0.7, 0)

else:
    col = (0, 0, 1)

r, g, b, = col

```

Here we decide what kind of colour we use to draw the temperature. It is based on the value selected with the slider.

```

bulbGrad = cairo.RadialGradient(-8, 250, 10,
                                -8, 250, 30)
bulbGrad.set_extend(cairo.EXTEND_REFLECT)
bulbGrad.add_color_stop_rgb(1, r, g, b)
bulbGrad.add_color_stop_rgba(0, r, g, b, 0)

cr.new_path()
cr.arc(0, 268, 9, 0, math.radians(360))
cr.set_source(bulbGrad)
cr.fill()

```

We use the radial gradient to fill the inside of the bulb. The bulb is the lower part of the scale.

```

scaleGrad = cairo.LinearGradient(0, 0, 5, 0)
scaleGrad.set_extend(cairo.EXTEND_REFLECT)

```

Previously, we have used a linear gradient to draw the background of the scale. This time the linear gradient draws the rectangle that indicates the temperature.

```

h, s, v = rgb_to_hsv(r, g, b)

r2, g2, b2 = hsv_to_rgb(h, s - t.SATURATION_DROP, v)

```

We use the HSV colour model. This model is better suited for humans. We decrease the saturation of the colour used for the temperature.

```

factor = self.tmpr-self.m_min
factor = (factor/self.m_max)-self.m_min

temp = t.SCALES_HEIGHT * factor
height = temp + t.TEMP_OFFSET

```

We compute the height of the temperature rectangle. The actual value in range 0..80 has to be transformed to fit the height of the scale. The `OFFSET` is the distance between the first line of the scale and the ellipse in the bulb.

```

cr.set_source(scaleGrad)
cr.rectangle(t.TEMP_X, t.SCALES_START +
            t.TEMP_OFFSET - height, t.TEMP_W, height)
cr.fill()

```

We draw the temperature rectangle with the linear gradient filling.

```

def OnSliderScroll(self, e):

```

```

"""Retrieves the value of the slider and sets it
    to the custom widget variable."""

obj = e.GetEventObject()
val = obj.GetValue()

self.cw.SetTemperature(val)
self.cw.Refresh()

```

In the `OnSliderScroll` method, we determine the currently selected value by the slider widget. We set the value to the `tmpr` variable of the Thermometer. The Thermometer is redrawn to reflect the changes.

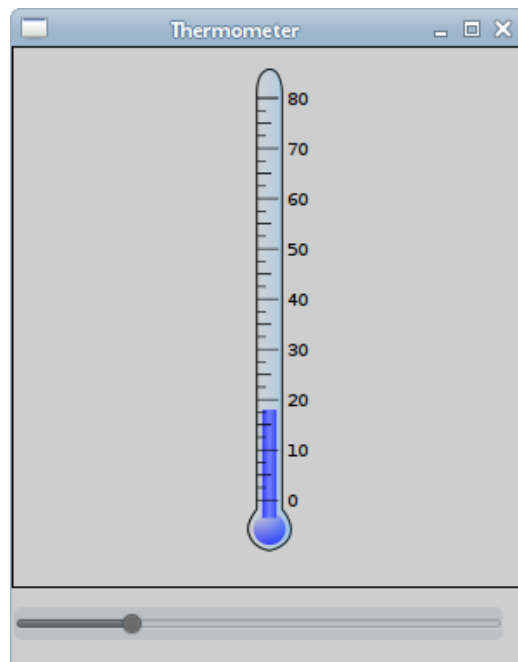


Figure 3.2: Custom Thermometer widget

Figure 3.2 shows a custom Thermometer widget.

3.3 ColourWheel widget

The ColourWheel widget is used to pick up a colour. A colour circle is an old concept. In the 17th century Isaac Newton has developed his own circular colour diagram.

There are three widely used colour models: RGB, CMYK and HSV. The RGB (Red, Green, Blue) model is used for the most part by computers. The CMYK (Cyan, Magenta, Yellow, Key) is a colour model popular in colour printing. Our custom widget is based on the HSV model.

The HSV (Hue, Saturation, Value) colour model was developed to have a colour scheme more intuitive for humans. It is also often called HSB (B for brightness). It defines a colour space in three terms. The hue is the colour type

(such as red, green or blue). It ranges from 0 to 360 degrees. The saturation of the colour ranges from 0 to 100%. It is the purity of the colour. The lower the saturation of a colour, the more grayness is present and the more faded the colour appears. The value/brightness of the colour ranges from 0 to 100%. It is the lightness of the colour. It tells us how distant our colour is from the black.

In our ColourWheel widget we have a colour circle. We pick a particular colour with a mouse pointer. The angle at which the colour was picked is the hue. The distance from the center of the circle is the saturation. The value is chosen in a separate indicator. There is a handle for selecting the value/brightness. The chosen colour is displayed in a separate rectangle. The RGB and the HSV values are shown using static text widgets. The RGB value is also present in the hexadecimal notation.

Listing 3.3: ColourWheel widget

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a custom
ColourWheel widget.

Author: Jan Bodnar
Website: zetcode.com
"""

import math
import sys
import wx
import cairo
from colorsys import hsv_to_rgb
import wx.lib.wxcairo

class cv:
    """Stores constants to avoid global values."""

    MARK_OFFSETX = 5
    MARK_OFFSETY = 2
    MARK_WIDTH = 30
    MARK_HEIGHT = 4

    RADIUS = 101.0

    BRIGHTNESS_X = 230
    BRIGHTNESS_Y = 30
    BRIGHTNESS_W = 20
    BRIGHTNESS_H = 150

class Wheel(wx.Panel):

    def __init__(self, parent, pos):
        super(Wheel, self).__init__(parent, pos=pos,
                                     size=(480, 250), style=wx.BORDER_THEME)

        self.InitUI()
```



```

def InitUI(self):
    """Initiates the Wheel custom control."""

    self.SetDoubleBuffered(True)
    self.col_pnl = wx.Panel(self, pos=(280, 20),
                             size=(100, 50))

    self.markx = cv.RADIUS
    self.marky = cv.RADIUS

    self.markx2 = 0
    self.marky2 = cv.BRIGHTNESS_Y

    # h 0..360; s 0..1; v 0..1
    self.hsv_store = [0.0, 0.0, 1.0]

    self.Bind(wx.EVT_PAINT, self.OnPaint)
    self.Bind(wx.EVT_LEFT_DOWN, self.OnLeftDown)
    self.Bind(wx.EVT_MOTION, self.OnMouseMove)

    self.sr = wx.StaticText(self, label='Red: 255',
                             pos=(280, 110))
    self.sg = wx.StaticText(self, label='Green: 255',
                             pos=(280, 130))
    self.sb = wx.StaticText(self, label='Blue: 255',
                             pos=(280, 150))

    self.sh = wx.StaticText(self, label='Hue: 0',
                             pos=(380, 110))
    self.ss = wx.StaticText(self, label='Saturation: 0',
                             pos=(380, 130))
    self.sv = wx.StaticText(self, label='Value: 100',
                             pos=(380, 150))

    self.shex = wx.StaticText(self, label='HEX: ffffff',
                               pos=(280, 190))

    self.whi = self.LoadWheelImage()
    self.bri = self.CreateBrightnessImg(1.0, 1.0, 1.0)

def OnPaint(self, e):
    """Draws objects on the panel."""

    dc = wx.PaintDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)

    self.DrawWheelImage(cr, self.whi)
    self.DrawBrightnessSelector(cr, self.bri)
    self.DrawRectangleMarker(cr)

def OnMouseMove(self, e):
    """Reacts to mouse move events over the wheel and
    brightness selector."""

    dc = wx.ClientDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)

    if e.LeftIsDown():
        x, y = e.GetX(), e.GetY()

```

```

        cr.arc(0, 0, cv.RADIUS, 0, math.radians(360))

        if cr.in_fill(x-cv.RADIUS, y-cv.RADIUS):

            self.ColourChanged(x, y)

            self.markx = x
            self.marky = y
            self.Refresh()

        cr.new_path()
        cr.rectangle(cv.BRIGHTNESS_X-5, cv.BRIGHTNESS_Y,
                     cv.BRIGHTNESS_W+2*5, cv.BRIGHTNESS_H)

        if cr.in_fill(x, y):

            val = 1-(y-cv.BRIGHTNESS_Y)/cv.BRIGHTNESS_H

            self.BrightnessChanged(val)

            self.markx2 = x
            self.marky2 = y
            self.Refresh()

def OnLeftDown(self, e):
    """Reacts to mouse press events over the wheel and
    brightness selector."""

    x, y = e.GetX(), e.GetY()

    dc = wx.ClientDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)
    cr.arc(0, 0, cv.RADIUS, 0, math.radians(360))

    if cr.in_fill(x-cv.RADIUS, y-cv.RADIUS):

        self.markx = x
        self.marky = y

        self.ColourChanged(x, y)
        self.Refresh()

    cr.new_path()
    cr.rectangle(cv.BRIGHTNESS_X, cv.BRIGHTNESS_Y,
                 cv.BRIGHTNESS_W, cv.BRIGHTNESS_H)

    if cr.in_fill(x, y):

        val = 1-(y-cv.BRIGHTNESS_Y)/cv.BRIGHTNESS_H

        self.BrightnessChanged(val)
        self.markx2 = x
        self.marky2 = y
        self.Refresh()

def CreateBrightnessImg(self, r, g, b):
    """Creates new brightness image. Takes r, g, b
    values from 0..1."""

```

```

bmp = wx.Bitmap(cv.BRIGHTNESS_W, cv.BRIGHTNESS_H, 24)
dc = wx.MemoryDC()
dc.SelectObject(bmp)

cr = wx.lib.wxcairo.ContextFromDC(dc)
cr.rectangle(0, 0, cv.BRIGHTNESS_W,
             cv.BRIGHTNESS_H)
lg = cairo.LinearGradient(0, 0, cv.BRIGHTNESS_W,
                          cv.BRIGHTNESS_H)
lg.add_color_stop_rgb(0, r, g, b)
lg.add_color_stop_rgb(1, 0, 0, 0)
cr.set_source(lg)
cr.fill()

dc.SelectObject(wx.NullBitmap)

return bmp

def BrightnessChanged(self, val):
    """Stores new brightness value. Draws new colour
    on the colour panel and updates the necessary
    labels."""

    h, s, v = self.hsv_store
    v = val
    self.hsv_store[2] = v

    r, g, b = hsv_to_rgb(h/360, s, v)

    col = wx.Colour(int(r*255), int(g*255), int(b*255))
    self.col_pnl.SetBackgroundColour(col)
    self.UpdateLabelsFromBrightnessChange(r, g, b)

def UpdateLabelsFromBrightnessChange(self, r, g, b):
    """Updates labels when the brightness value
    changes. The hue and saturation are not affected."""

    self.sv.SetLabelText('Value: %d' %
                        (self.hsv_store[2]*100))

    self.sr.SetLabelText('Red: %d' % (r*255))
    self.sg.SetLabelText('Green: %d' % (g*255))
    self.sb.SetLabelText('Blue: %d' % (b*255))

    self.shex.SetLabelText('HEX: %02x%02x%02x' %
                        (int(r*255), int(g*255), int(b*255)))

def ColourChanged(self, x, y):
    """Called in reaction to newly selected colour on
    the wheel. Stores new hue and saturation values,
    updates all affected labels and creates a new
    brightness image."""

    a = x - cv.RADIUS
    b = y - cv.RADIUS

    hue, s = self.CalculateHueSaturation(a, b)

    self.hsv_store[0] = hue

```

```

self.hsv_store[1] = s/cv.RADIUS

# hsv_to_rgb takes h, s, v from 0..1
r, g, b = hsv_to_rgb(hue/360.0, s/cv.RADIUS,
    self.hsv_store[2])

col = wx.Colour(int(r*255), int(g*255), int(b*255))
self.col_pnl.SetBackgroundColour(col)

self.UpdateLabelsFromColourChange(r, g, b, hue, s)

# this is for brightness image
r, g, b = hsv_to_rgb(hue/360.0, s/cv.RADIUS, 1.0)
self.bri = self.CreateBrightnessImg(r, g, b)

def CalculateHueSaturation(self, a, b):
    """ Calculates hue and saturation.
        Returns hue: 0..360; s 0..100. """

    s = math.sqrt(pow(a, 2) + pow(b, 2))

    h = math.atan2(b, a)
    hue = math.degrees(h)

    if hue < 0:
        hue = abs(hue)
    elif hue > 0:
        hue = 360 - hue

    return hue, s

def UpdateLabelsFromColourChange(self, r, g, b, hue, s):
    """Updates labels affected by a colour change.
        The brightness is not affected. """

    self.sr.SetLabelText('Red: %d' % (r*255))
    self.sg.SetLabelText('Green: %d' % (g*255))
    self.sb.SetLabelText('Blue: %d' % (b*255))

    self.sh.SetLabelText('Hue: %d' % hue)
    self.ss.SetLabelText('Saturation: %d' % s)

    self.shex.SetLabelText('HEX: %02x%02x%02x' %
        (int(r*255), int(g*255), int(b*255)))

def LoadWheelImage(self):
    """Loads the colour wheel image. """

    try:
        self.img = wx.Bitmap("wheel.png",
            wx.BITMAP_TYPE_PNG)

    except Exception as e:

        print(e.message)
        sys.exit(1)

    return self.img

```

```

def DrawWheelImage(self, cr, bmp):
    """Draws the colour wheel image and the circle
    marker on the panel."""

    imgs = wx.lib.wxcairo.ImageSurfaceFromBitmap(bmp)

    cr2 = cairo.Context(imgs)

    cr2.arc(self.markx, self.marky, 3, 0,
            math.radians(360))
    cr2.stroke()

    cr2.set_source_rgba(0.0, 0.0, 0.0, 0.0)
    cr.set_source_surface(imgs, 0, 0)
    cr.paint()

def DrawBrightnessSelector(self, cr, bmp):
    """Draws the brightness image on the panel."""

    imgs = wx.lib.wxcairo.ImageSurfaceFromBitmap(bmp)

    cr.set_source_surface(imgs, cv.BRIGHTNESS_X,
                        cv.BRIGHTNESS_Y)
    cr.paint()

def DrawRectangleMarker(self, cr):
    """Draws the rectangle marker over the
    brightness image."""

    cr.set_line_width(1)
    cr.set_source_rgb(0, 0, 0)
    cr.rectangle(cv.BRIGHTNESS_X - cv.MARK_OFFSETX,
                self.marky2 - cv.MARK_OFFSETY, cv.MARK_WIDTH,
                cv.MARK_HEIGHT)
    cr.stroke()

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.SetSize((500, 290))
        self.SetTitle("ColourWheel")
        self.Centre()

    def InitUI(self):

        pnl = wx.Panel(self)
        self.cw = Wheel(pnl, pos=(5, 5))

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

```

```
if __name__ == "__main__":
    main()
```

In a ColourWheel widget the colour is picked by a mouse pointer. There is a small circular marker over the currently selected hue and saturation values. The brightness is controlled by a handle in an adjacent selector.

```
from colorsys import hsv_to_rgb
```

From the `colorsys` Python module we use the `hsv_to_rgb` method to calculate the RGB values from the HSV ones.

```
MARK_OFFSETX = 5
MARK_OFFSETY = 2
MARK_WIDTH = 30
MARK_HEIGHT = 4
```

These constants are used for the marker handle which selects the current brightness value.

```
BRIGHTNESS_X = 230
BRIGHTNESS_Y = 30
BRIGHTNESS_W = 20
BRIGHTNESS_H = 150
```

The above four constants store the x, y coordinates and the width and height of the brightness selector.

```
self.col_pnl = wx.Panel(self, pos=(280, 20),
                        size=(100, 50))
```

This is a panel in which we show the currently selected colour.

```
self.markx = cv.RADIUS
self.marky = cv.RADIUS

self.markx2 = 0
self.marky2 = cv.BRIGHTNESS_Y
```

These four variables hold the x, y coordinates of objects that we call markers. The first pair stores the coordinates of a small circle drawn over the colour circle. The second pair stores the coordinates of the handle that is moved to select the brightness value.

```
# h 0..360; s 0..1; v 0..1
self.hsv_store = [0.0, 0.0, 1.0]
```

We store the current HSV values in the `hsv_store` list. The hue is an integer between 0 and 360. The saturation and value are floating point numbers between 0 and 1.

```
self.sr = wx.StaticText(self, label='Red: 255',
                        pos=(280, 110))
self.sg = wx.StaticText(self, label='Green: 255',
```

```

        pos=(280, 130))
    ...

```

Colour values are displayed with static text widgets. The constructors have some default values, which are later modified.

```

self.whi = self.LoadWheelImage()

```

This method loads the colour wheel image. Drawing our own colour wheel image would be very expensive. Instead of that, we have a precreated image that is drawn on the window.

```

self.bri = self.CreateBrightnessImg(1.0, 1.0, 1.0)

```

A brightness image is created. It is the background of the value selector.

```

def OnPaint(self, e):
    """Draws objects on the panel."""

    dc = wx.PaintDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)

    self.DrawWheelImage(cr, self.whi)
    self.DrawBrightnessSelector(cr, self.bri)
    self.DrawRectangleMarker(cr)

```

In the `OnPaint` method the drawing is delegated into three methods. The `DrawWheelImage` draws the colour wheel image and its circular marker. The `DrawBrightnessSelector` draws the background of the brightness selector. The `DrawRectangleMarker` method draws the marker over the brightness selector.

```

def OnMouseMove(self, e):
    """Reacts to mouse move events over the wheel and
    brightness selector."""

    dc = wx.ClientDC(self)
    cr = wx.lib.wxcairo.ContextFromDC(dc)
    ...

```

In the `OnMouseMove` method, we react to mouse move events. These events are important for both the colour image and the value selector. We work on the client area, so we create a cairo drawing context from the client device context.

```

if e.LeftIsDown():
    x, y = e.GetX(), e.GetY()

    cr.arc(0, 0, cv.RADIUS, 0, math.radians(360))

    if cr.in_fill(x-cv.RADIUS, y-cv.RADIUS):

        self.ColourChanged(x, y)

        self.markx = x
        self.marky = y
        self.Refresh()

```

The above code is executed in reaction to a mouse move event with the left mouse button being pressed. First, we get the x, y coordinates of the mouse

pointer. We call the `arc` method of the cairo context. Note that we do not intend to draw here. Actual drawing is done with `stroke` or `fill` methods.

With the `arc` method we create a new path which is a circle exactly matching the colour wheel image. The path is used to determine whether we are within the bounds of the colour wheel image. The method for this is called `in_fill`. If we are inside the circle path, we call the `ColorChanged` method and update the variables which store the x, y coordinates of the mouse pointer. We redraw the client area.

```
cr.new_path()
cr.rectangle(cv.BRIGHTNESS_X-5, cv.BRIGHTNESS_Y,
             cv.BRIGHTNESS_W+2*5, cv.BRIGHTNESS_H)

if cr.in_fill(x, y):

    val = 1-(y-cv.BRIGHTNESS_Y)/cv.BRIGHTNESS_H

    self.BrightnessChanged(val)

    self.markx2 = x
    self.marky2 = y
    self.Refresh()
```

The `new_path` method drops the previous path (it was a circle) and starts a new one. Here we check if the mouse pointer is inside the brightness selector. If so, a new value is computed and the `BrightnessChanged` method is called. We store the new x, y coordinates of the marker and repaint the client area.

```
def OnLeftDown(self, e):
    """Reacts to mouse press events over the wheel and
        brightness selector."""
    ...
```

The `OnLeftDown` method reacts to mouse press events. It does practically the same as the previous one. The difference is that we only react to mouse press events.

```
def CreateBrightnessImg(self, r, g, b):
    '''Creates new brightness image. Takes r, g, b
        values from 0..1.'''
    ...
```

The `CreateBrightnessImg` creates a background image for the brightness selector.

```
bmp = wx.EmptyBitmap(cv.BRIGHTNESS_W,
                     cv.BRIGHTNESS_H, 24)
dc = wx.MemoryDC()
dc.SelectObject(bmp)
```

We create an empty image in memory; this makes the drawing faster. The third parameter of the `wx.EmptyBitmap` class is the colour depth. You may experience errors here, since the value is platform dependent. Value 24 works both on Linux and Windows. Changing the colour depth to 32 or to -1 (the default) works on Linux, but leads to errors on Windows.

```
cr = wx.lib.wxcairo.ContextFromDC(dc)
cr.rectangle(0, 0, cv.BRIGHTNESS_W,
```



```

        cv.BRIGHTNESS_H)
lg = cairo.LinearGradient(0, 0, cv.BRIGHTNESS_W,
        cv.BRIGHTNESS_H)
lg.add_color_stop_rgb(0, r, g, b)
lg.add_color_stop_rgb(1, 0, 0, 0)
cr.set_source(lg)
cr.fill()

```

First, we create a cairo drawing context from the memory image. We draw a rectangle and fill it with a linear gradient of a currently selected colour and a black colour. Remember that the third letter of the HSB colour scheme is the brightness/lightness of the colour. This value tells us how distant the colour is from the black. This distance is visually represented by a linear gradient.

```

def BrightnessChanged(self, val):
    """Stores new brightness value. Draws new colour
    on the colour panel and updates the necessary
    labels."""

    h, s, v = self.hsv_store
    v = val
    self.hsv_store[2] = v

    r, g, b = hsv_to_rgb(h/360, s, v)

    col = wx.Colour(int(r*255), int(g*255), int(b*255))
    self.col_pnl.SetBackgroundColour(col)
    self.UpdateLabelsFromBrightnessChange(r, g, b)

```

In the `BrightnessChanged` method, we store the new brightness value and calculate the RGB values with the `hsv_to_rgb` method. We set a new colour for the colour panel and update the labels that are affected by the brightness change.

```

def UpdateLabelsFromBrightnessChange(self, r, g, b):
    """Updates labels when the brightness value
    changes. The hue and saturation are not affected."""
    ...

```

In the `UpdateLabelsFromBrightnessChange` method we update the labels that are affected by the brightness value change. The hue and saturation labels are not altered.

```

def ColourChanged(self, x, y):
    """Called in reaction to newly selected colour on
    the wheel. Stores new hue and saturation values,
    updates all affected labels and creates a new
    brightness image."""
    ...

```

The `ColourChanged` method is called when a new colour is selected on the colour wheel image. The new hue and saturation values are calculated and stored in a variable. The affected labels are updated and a new brightness image is created.

```

def CalculateHueSaturation(self, a, b):
    """Calculates hue and saturation.
    Returns hue: 0..360; s 0..100."""
    ...

```

This method is called from the previously mentioned method. We calculate the hue and saturation values.

```
s = math.sqrt(pow(a, 2) + pow(b, 2))
```

The saturation is the distance from the mouse pointer to the center of the circle. Having two points, a right triangle can be formed. We have the two legs and using the Pythagoras theorem, we get the third side—the hypotenuse. The hypotenuse is the saturation value in our colour model.

```
h = math.atan2(b, a)
hue = math.degrees(h)
```

```
if hue < 0:
    hue = abs(hue)
elif hue > 0:
    hue = 360 - hue
```

We use the `atan2` method to get the theta angle of the triangle. The angle is returned in radians. The `degrees` method is used to convert the radians into degrees. The `atan2` method calculates the angle of a right triangle, so it returns an angle between pi and -pi. We adjust the angle to fit the angles of a circle. We get the hue value of the HSV colour model.

```
def UpdateLabelsFromColourChange(self, r, g, b, hue, s):
    """Updates labels affected by a colour change.
       The brightness is not affected."""

    self.sr.SetLabelText('Red: %d' % (r*255))
    self.sg.SetLabelText('Green: %d' % (g*255))
    self.sb.SetLabelText('Blue: %d' % (b*255))
    ...
```

Here, the labels are being updated. The `cairo` library works with colours on a 0..1 scale. We multiply the r, g, b values by 255 to get a more common scale. 0..255.

```
def DrawWheelImage(self, cr, bmp):
    """Draws the colour wheel image and the circle
       marker on the panel."""

    imgs = wx.lib.wxcairo.ImageSurfaceFromBitmap(bmp)

    cr2 = cairo.Context(imgs)
    cr2.arc(self.markx, self.marky, 3, 0,
            math.radians(360))
    cr2.stroke()

    cr2.set_source_rgba(0.0, 0.0, 0.0, 0.0)
    cr.set_source_surface(imgs, 0, 0)
    cr.paint()
```

In the `DrawWheelImage` method, we draw the colour wheel image with a small circular marker on the panel.

```
def DrawBrightnessSelector(self, cr, bmp):
    """Draws the brightness image on the panel."""
```

```

imgs = wx.lib.wxcairo.ImageSurfaceFromBitmap bmp)

cr.set_source_surface(imgs, cv.BRIGHTNESS_X,
    cv.BRIGHTNESS_Y)
cr.paint()

```

We have created a brightness image in the `CreateBrightnessImg` method. Here it is drawn on the panel.

```

def DrawRectangleMarker(self, cr):
    """Draws the rectangle marker over the
    brightness image."""

    cr.set_line_width(1)
    cr.set_source_rgb(0, 0, 0)
    cr.rectangle(cv.BRIGHTNESS_X - cv.MARK_OFFSETX,
        self.marky2 - cv.MARK_OFFSETY, cv.MARK_WIDTH,
        cv.MARK_HEIGHT)
    cr.stroke()

```

A marker over the brightness image is drawn with the `DrawRectangleMarker` method.

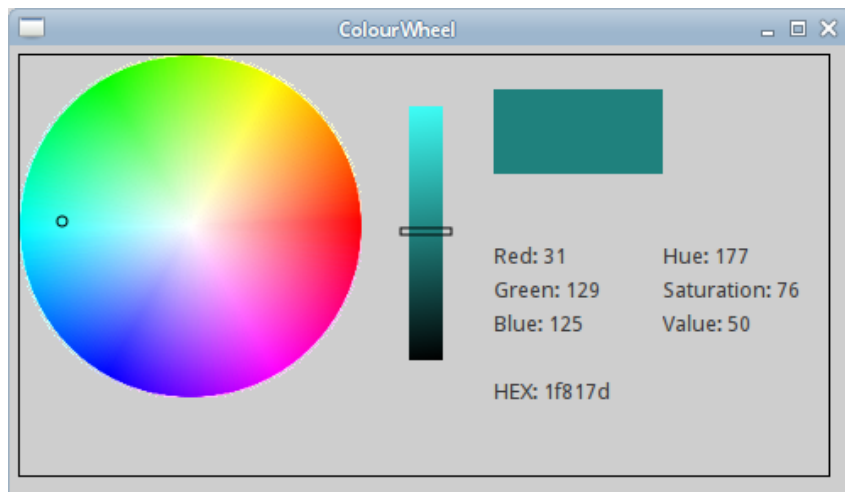


Figure 3.3: Custom ColourWheel widget

Figure 3.3 shows a custom ColourWheel widget.

Part II

Advanced widgets

Chapter 4

wx.TextCtrl

A `wx.TextCtrl` is a text control widget. It allows text to be displayed and edited. It may be single line or multi-line. The multi-line text controls always store the text as a sequence of lines separated by `\n` characters.

The `wx.TextCtrl` is a simple text control widget. It provides some basic functionality for text editing. We may do some text selections or elementary text styling. `wx.richtext.RichTextCtrl` and `wx.stc.StyledTextCtrl` are widgets with more advanced text features.

4.1 Simple example

In the first example, we will do two things. We build a menu with menu items that will clear all text and toggle the ability to edit the text control.

Listing 4.1: Simple example

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

This example shows how to toggle editability
of the wx.TextCtrl and to clear all text.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):
```

```

        self.CreateMenuBar()

        self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE)

        self.SetSize((750, 450))
        self.SetTitle('TextCtrl')
        self.Centre()

def CreateMenuBar(self):
    ID_TEXT_CLEAR = wx.NewIdRef()
    ID_TEXT_READONLY = wx.NewIdRef()

    mb = wx.MenuBar()

    eMenu = wx.Menu()
    eMenu.Append(ID_TEXT_CLEAR, 'Clear Text')
    eMenu.AppendCheckItem(ID_TEXT_READONLY, 'Readonly')

    mb.Append(eMenu, '&Edit')
    self.SetMenuBar(mb)

    self.Bind(wx.EVT_MENU, self.OnTextClear,
               id=ID_TEXT_CLEAR)
    self.Bind(wx.EVT_MENU, self.SetReadOnly,
               id=ID_TEXT_READONLY)

    self.Bind(wx.EVT_UPDATE_UI, self.CheckMenuItemClear,
               id=ID_TEXT_CLEAR)

def CheckMenuItemClear(self, e):
    e.Enable(not self.te.IsEmpty())

    if not self.te.IsEditable():
        e.Enable(False)

def OnTextClear(self, e):
    self.te.Clear()

def SetReadOnly(self, e):
    if e.IsChecked():
        self.te.SetEditable(False)
    else:
        self.te.SetEditable(True)

def main():
    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":

```

```
main()
```

There is a menubar and a `wx.TextCtrl` widget in the example.

```
self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE)
```

A text control takes the whole area of the window. The `wx.TE_MULTILINE` style creates a multi-line text control.

```
def CreateMenuBar(self):  
    ID_TEXT_CLEAR = wx.NewIdRef()  
    ID_TEXT_READONLY = wx.NewIdRef()  
    ...
```

Menubar creation is delegated to the `CreateMenuBar` method. We create two ids with `wx.NewIdRef` for the two menu items.

```
mb = wx.MenuBar()  
  
eMenu = wx.Menu()  
eMenu.Append(ID_TEXT_CLEAR, 'Clear Text')  
eMenu.AppendCheckItem(ID_TEXT_READONLY, 'Readonly')  
  
mb.Append(eMenu, '&Edit')  
self.SetMenuBar(mb)
```

A menubar with two menu items is created.

```
self.Bind(wx.EVT_MENU, self.OnTextClear,  
          id=ID_TEXT_CLEAR)  
self.Bind(wx.EVT_MENU, self.SetReadOnly,  
          id=ID_TEXT_READONLY)  
  
self.Bind(wx.EVT_UPDATE_UI, self.CheckMenuItemClear,  
          id=ID_TEXT_CLEAR)
```

Events are bound to their methods. The `wx.EVT_UPDATE_UI` is a specific event that helps us update user interface elements. Just before a menu pops up, the event is called and we can update the UI elements accordingly. In our case, we want to disable the text clear menu item if the text control is in a readonly mode or it is empty.

```
def CheckMenuItemClear(self, e):  
    e.Enable(not self.te.IsEmpty())  
  
    if not self.te.IsEditable():  
        e.Enable(False)
```

The text clear menu item is disabled when the text control is empty and when the text control is in the readonly mode. The `Enable` method enables or disables the UI element.

```
def OnTextClear(self, e):  
    self.te.Clear()
```

The `Clear` method clears all text in the text control.

```
def SetReadOnly(self, e):  
    if e.IsChecked():  
        self.te.SetEditable(False)  
    else:  
        self.te.SetEditable(True)
```

The `SetEditable` method toggles the readonly mode of the text control.

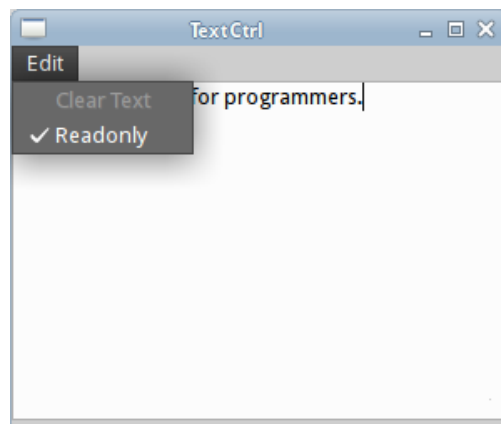


Figure 4.1: wx.TextCtrl simple example

Figure 4.1 shows a screenshot of a `wx.TextCtrl` simple example.

4.2 Lines and columns

The second example shows the current character's line and column number in the the statusbar.

Listing 4.2: Lines and columns

```
#!/usr/bin/env python  
  
"""  
ZetCode Advanced wxPython tutorial  
  
In this code example we show the current  
line and column number in the statusbar.  
  
Author: Jan Bodnar  
Website: zetcode.com  
"""  
  
import wx  
  
class Example(wx.Frame):  
    def __init__(self, *args, **kw):  
        super(Example, self).__init__(*args, **kw)
```



```

        self.InitUI()

def InitUI(self):

    sb = self.CreateStatusBar()
    sb.SetStatusText("Line: 1, Col: 1")

    self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE)

    self.te.Bind(wx.EVT_TEXT, self.OnTextChanged)
    self.te.Bind(wx.EVT_LEFT_UP, self.OnMouseLeftUp)
    self.te.Bind(wx.EVT_KEY_UP, self.OnKeyUp)

    self.SetSize((700, 450))
    self.SetTitle('TextCtrl')
    self.Centre()

def OnMouseLeftUp(self, e):

    self.UpdateStatusbar()

def OnKeyUp(self, e):

    key = e.GetKeyCode()

    if key in (wx.WXK_LEFT, wx.WXK_RIGHT,
               wx.WXK_DOWN, wx.WXK_UP, wx.WXK_HOME, wx.WXK_END,
               wx.WXK_PAGEUP, wx.WXK_PAGEDOWN):

        self.UpdateStatusbar()

def OnTextChanged(self, e):

    self.UpdateStatusbar()

def UpdateStatusbar(self):

    sb = self.GetStatusBar()
    ip = self.te.GetInsertionPoint()

    print(self.te.PositionToXY(ip))

    _, col, ln = self.te.PositionToXY(ip)

    txt = f"Line: {ln + 1}, Col: {col + 1}"
    sb.SetStatusText(txt)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

```

```
if __name__ == "__main__":
    main()
```

There is a text control and a statusbar in the example. The current line and column numbers are shown in the statusbar.

```
sb = self.CreateStatusBar()
sb.SetStatusText("Line: 1, Col: 1")
```

A statusbar is created with the `CreateStatusBar` method. The initial line and column numbers are set to the statusbar; the numbers start from 1.

```
self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE)
```

A multi-line text control is created.

```
self.te.Bind(wx.EVT_TEXT, self.OnTextChanged)
self.te.Bind(wx.EVT_LEFT_UP, self.OnMouseLeftUp)
self.te.Bind(wx.EVT_KEY_UP, self.OnKeyUp)
```

Three events are bound to methods. The `wx.EVT_TEXT` event is generated when the text changes in the text control. The `wx.EVT_LEFT_UP` is generated when the mouse left button is released and the `wx.EVT_KEY_UP` when the key is released. All these events influence the position of the caret in the text control.

```
def OnMouseLeftUp(self, e):

    self.UpdateStatusbar()
```

We can always change the current position of the caret with the mouse. The `OnMouseLeftUp` method is called after the left mouse button is released. The method calls the `UpdateStatusbar` method.

```
def OnKeyUp(self, e):

    key = e.GetKeyCode()

    if key in (wx.WXK_LEFT, wx.WXK_RIGHT,
               wx.WXK_DOWN, wx.WXK_UP, wx.WXK_HOME, wx.WXK_END,
               wx.WXK_PAGEUP, wx.WXK_PAGEDOWN):

        self.UpdateStatusbar()
```

This method is called when we release a key button on the keyboard. The `UpdateStatusbar` method is called for the cursor keys, page down, page up, home and end keys.

```
def OnTextChanged(self, e):

    self.UpdateStatusbar()
```

When the text changes, either when we write some text or delete some text, the `UpdateStatusbar` method is called.

```
def UpdateStatusbar(self):
```

```

sb = self.GetStatusBar()
ip = self.te.GetInsertionPoint()

_, col, ln = self.te.PositionToXY(ip)

txt = f"Line: {ln + 1}, Col: {col + 1}"
sb.SetStatusText(txt)

```

The `GetInsertionPoint` method gets the insertion point. It is a zero based index of the character position. The `PositionToXY` method converts the given insertion point to (bool, x, y) tuple. We do not need the first value; therefore, we use the discard. The x and y numbers are shown on the statusbar.

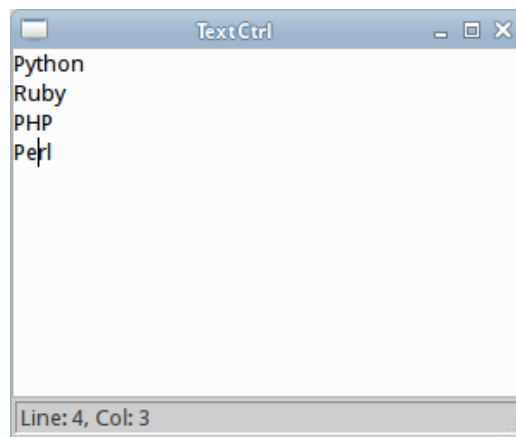


Figure 4.2: Lines and columns

Figure 4.2 shows a text control, with the current line and column number shown in the statusbar.

4.3 Selections

Text can be selected with a mouse or with keyboard keys. Cursors keys with either control or shift keys hold. The colour of the selected text and its background is automatically changed.

Listing 4.3: Selections

```

#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we work with text selections.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

```

```

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE)

        self.te.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)

        self.SetSize((750, 450))
        self.SetTitle('TextCtrl')
        self.Centre()

    def OnKeyDown(self, e):

        key = e.GetKeyCode()

        if e.ControlDown() and key == ord('T'):

            print(self.te.GetSelection())
            print(self.te.GetStringSelection())

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

In the example, we print the current selection indexes and the selected text to the terminal. The information is printed when we press the Ctrl+T key combination.

```
self.te.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
```

We bind the `wx.EVT_KEY_DOWN` event to the `OnKeyDown` method.

```

def OnKeyDown(self, e):

    key = e.GetKeyCode()

    if e.ControlDown() and key == ord('T'):

        print(self.te.GetSelection())
        print(self.te.GetStringSelection())

```

Upon pressing the Ctrl+T shortcut, two methods are called. The first method, the `GetSelection`, returns the indexes of the current selection span. The second

method, the `GetStringSelection`, returns the selected text in the text control. The values are printed to the terminal.

4.4 Text search example

In the next example we search for text. The background colour of the text that was found will change. To change the style of the text, we use the `wx.TextAttr` which represents the character and paragraph attributes, or style, for a range of text in a `wx.TextCtrl`. Remember that this widget allows only rudimentary work with text styles. On Windows we need to provide an additional `wx.TE_RICH` window style; it is ignored on other platforms.

Listing 4.4: Text search

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

This example we search for text
in the wx.TextCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        vbox = wx.BoxSizer(wx.VERTICAL)

        self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE |
                               wx.TE_RICH)
        vbox.Add(self.te, proportion=1, flag=wx.EXPAND)

        self.StoreColours()

        txt = "Bird\nCat\n\nDog\nMouse\tCat\tCamel"
        self.te.AppendText(txt)

        self.tinput = wx.TextCtrl(self)
        self.tinput.SetMinSize((120, -1))
        self.tinput.SetFocus()

        sbtn = wx.Button(self, label="Search")
        sbtn.Bind(wx.EVT_BUTTON, self.OnSearchText)

        rbtn = wx.Button(self, label="Reset")
        rbtn.Bind(wx.EVT_BUTTON, self.OnReset)
```

```

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add(self.tinput)
        hbox.Add(sbtn)
        hbox.Add(rbtn)

        vbox.Add(hbox)

        self.SetSizer(vbox)

        self.SetSize((750, 450))
        self.SetTitle('TextCtrl')
        self.Centre()

def StoreColours(self):

    bc = self.te.GetBackgroundColour()
    fc = self.te.GetForegroundColour()

    self.bcol = wx.Colour(bc[0], bc[1], bc[2], bc[3])
    self.fcol = wx.Colour(fc[0], fc[1], fc[2], fc[3])

def OnSearchText(self, e):

    text = self.te.GetValue()
    word = self.tinput.GetValue()
    lng = len(word)

    self.ResetStyle()

    atr = wx.TextAttr(wx.BLACK, wx.LIGHT_GREY)

    found = text.find(word)

    while found > -1:

        self.te.SetStyle(found, found+lng, atr)

        found = text.find(word, found + 1)

def OnReset(self, e):

    self.ResetStyle()

def ResetStyle(self):

    self.te.SetStyle(0, -1, wx.TextAttr(self.fcol, self.bcol))

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

There is a multi-line `wx.TextCtrl` widget in the example. It takes the most of the space in the layout. Below it, there are two other widgets. A single-line text control and a search button.

```
vbox = wx.BoxSizer(wx.VERTICAL)
...
hbox = wx.BoxSizer(wx.HORIZONTAL)
```

The layout of the example is a bit more complicated, therefore we use box sizers.

```
self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE|
                      wx.TE_RICH)
```

The `wx.TE_RICH` style enables styles under Windows.

```
txt = "Bird\nCat\n\nDog\nMouse\tCat\tCamel"
self.te.AppendText(txt)
```

This text is appended to the main text control.

```
self.tinput = wx.TextCtrl(self)
self.tinput.SetMinSize((120, -1))
self.tinput.SetFocus()
```

The single-line search text control is created. We set a minimal size for this widget. Also, it has initially the keyboard focus.

```
def OnKeyDown(self, e):
    key = e.GetKeyCode()

    if key == wx.WXK_ESCAPE:
        self.te.SetStyle(0, -1, wx.TextAttr(wx.BLACK,
                                              wx.WHITE))
```

The Esc key sets the style of the text to the default. The text is not highlighted anymore. The `SetStyle` method changes the style of the text for the given range. The 0, -1 range stands for all text.

```
def OnSearchText(self, e):
    text = self.te.GetValue()
    word = self.tinput.GetValue()
    lng = len(word)
    ...
```

In the `OnSearchText` method, we do the text searching. The first two lines get the text from the both text controls. The length of the search term is computed.

```
self.te.SetStyle(0, -1, wx.TextAttr(wx.BLACK,
                                     wx.WHITE))
```

A new search removes the highlight of a previous search.

```
atr = wx.TextAttr(wx.BLACK, wx.LIGHT_GREY)
```

This text attribute is used for the words that were found. The first colour is the foreground colour. The second is the background colour.

```
found = text.find(word)
```

We use the Python built-in string `find` method to search for the text. It returns the lowest index of the located substring.

```
while found > -1:
    self.te.SetStyle(found, found+lng, atr)
    found = text.find(word, found + 1)
```

There might be multiple hits in the text control. Therefore, we use a while loop to search for all occurrences.

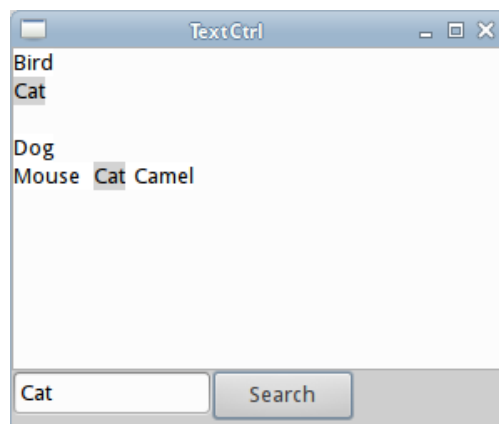


Figure 4.3: Text search example

Figure 4.3 shows highlighted text for the term that was found in the text control.

4.5 Load, save example

The final example is a skeleton of a text editor. We create a functionality to edit, load and save simple text files. The information whether the file was opened, saved or modified is displayed in the statusbar.

Listing 4.5: Load & save example

```
#!/usr/bin/env python

'''
ZetCode Advanced wxPython tutorial

In this program we will load and save
text data.

Author: Jan Bodnar
Website: zetcode.com
'''

import wx
import os
```



```

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateMenuBar()
        self.BuildStatusBar()

        self.te = wx.TextCtrl(self, style=wx.TE_MULTILINE)
        self.te.Bind(wx.EVT_TEXT, self.OnTextChanged)
        self.last_name_saved = ''

        self.SetSize((750, 450))
        self.SetTitle('TextCtrl')
        self.Centre()

    def BuildStatusBar(self):

        sb = self.CreateStatusBar(2)
        sb.SetStatusWidths([-3, -1])
        sb.SetStatusText("Ready")

    def CreateMenuBar(self):

        ID_MENU_OPEN = wx.NewIdRef()
        ID_MENU_SAVE = wx.NewIdRef()
        ID_MENU_SAVEAS = wx.NewIdRef()

        mb = wx.MenuBar()

        self.fMenu = wx.Menu()
        self.fMenu.Append(ID_MENU_OPEN, '&Open...')
        self.fMenu.Append(ID_MENU_SAVE, '&Save')
        self.fMenu.Append(ID_MENU_SAVEAS, 'Save &as...')

        mb.Append(self.fMenu, '&File')

        self.SetMenuBar(mb)

        self.Bind(wx.EVT_MENU, self.OnOpenFile,
                  id=ID_MENU_OPEN)
        self.Bind(wx.EVT_MENU, self.OnSaveFile,
                  id=ID_MENU_SAVE)
        self.Bind(wx.EVT_MENU, self.OnSaveAsFile,
                  id=ID_MENU_SAVEAS)

    def OnTextChanged(self, e):

        sb = self.GetStatusBar()
        sb.SetStatusText(' modified', 1)

    def OnOpenFile(self, e):

```

```

if self.te.IsModified():
    dlg = wx.MessageDialog(self, 'Save changes?',
        '', wx.YES_NO | wx.YES_DEFAULT
        | wx.CANCEL | wx.ICON_QUESTION)

    val = dlg.ShowModal()

    if val == wx.ID_YES:
        self.OnSaveFile(e)
        self.OpenReadFile()

    elif val == wx.ID_CANCEL:
        dlg.Destroy()

    else:
        self.OpenReadFile()
else:
    self.OpenReadFile()

def OpenReadFile(self):

    wcd = 'All files (*)|*|Editor files (*.ef)|*.ef'

    dirname = os.getcwd()

    od = wx.FileDialog(self, message='Choose a file',
        defaultDir=dirname, defaultFile='',
        wildcard=wcd, style=wx.FD_OPEN|wx.FD_CHANGE_DIR)

    if od.ShowModal() == wx.ID_OK:
        path = od.GetPath()

        fname = None

        try:
            fname = open(path, 'r')
            text = fname.read()

            if self.te.GetLastPosition():
                self.te.Clear()

            self.te.WriteText(text)
            self.last_name_saved = path
            self.te.SetModified(False)

            sb = self.GetStatusBar()
            stat = os.path.basename(path) + ' opened'
            sb.SetStatusText(stat, 0)
            sb.SetStatusText('', 1)

        except IOError as e:

            dlg = wx.MessageDialog(self,
                'Error opening file\n' + str(e))
            dlg.ShowModal()
            dlg.Destroy()

        except UnicodeDecodeError as e:

            dlg = wx.MessageDialog(self,
                'Error opening file\n' + str(e))

```

```

        dlg.ShowModal()
        dlg.Destroy()

    finally:

        if fname:
            fname.close()

    od.Destroy()

def OnSaveFile(self, e):

    lns = self.last_name_saved

    if lns:

        fname = None

        try:

            fname = open(lns, 'w')
            text = self.te.GetValue()
            fname.write(text)

            sb = self.GetStatusBar()
            stat = os.path.basename(lns) + ' saved'
            sb.SetStatusText(stat, 0)
            sb.SetStatusText('', 1)
            self.te.SetModified(False)

        except IOError as e:

            dlg = wx.MessageDialog(self,
                                   'Error saving file\n' + str(e))
            dlg.ShowModal()
            dlg.Destroy()

        except UnicodeEncodeError as e:

            dlg = wx.MessageDialog(self,
                                   'Error saving file\n' + str(e))
            dlg.ShowModal()
            dlg.Destroy()

    finally:

        if fname:
            fname.close()

    else:
        self.OnSaveAsFile(e)

def OnSaveAsFile(self, e):

    wcd='All files(*)|*|Editor files (*.ef)|*.ef'

    directory = os.getcwd()

    sd = wx.FileDialog(self, message='Save file as...',
                      defaultDir=directory, defaultFile='',
                      wildcard=wcd,
                      style=wx.FD_SAVE|wx.FD_OVERWRITE_PROMPT)

```

```

        if sd.ShowModal() == wx.ID_OK:

            path = sd.GetPath()

            fname = None

            try:
                fname = open(path, 'w')
                text = self.te.GetValue()
                fname.write(text)

                sb = self.GetStatusBar()
                stat = os.path.basename(path) + ' saved'
                sb.SetStatusText(stat, 0)
                sb.SetStatusText('', 1)
                self.te.SetModified(False)

            except IOError as e:

                dlg = wx.MessageDialog(self,
                                        'Error saving file\n' + str(e))
                dlg.ShowModal()
                dlg.Destroy()

            except UnicodeEncodeError as e:

                dlg = wx.MessageDialog(self,
                                        'Error saving file\n' + str(e))
                dlg.ShowModal()
                dlg.Destroy()

            finally:

                if fname:
                    fname.close()

        sd.Destroy()

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

In this example, we have a menubar, a statusbar and a `wx.TextCtrl` widget. The menubar has three items: an item for opening a file, saving a file, and saving a file under a different name. The statusbar shows the information about the current file being edited.

```
self.last_name_saved = ''
```

We store the name of the last saved file in the `last_name_saved` variable.

```
def BuildStatusBar(self):
```

```

sb = self.CreateStatusBar(2)
sb.SetStatusWidths([-3, -1])
sb.SetStatusText("Ready")

```

This method builds a statusbar. The statusbar has two parts. In the first part we display the name of the file. In the second part we show the modified flag.

```

self.fMenu = wx.Menu()
self.fMenu.Append(ID_MENU_OPEN, '&Open...')
self.fMenu.Append(ID_MENU_SAVE, '&Save')
self.fMenu.Append(ID_MENU_SAVEAS, 'Save &as...')

```

The menubar consists of three menu items. They open a file, save a file, and save a file under a different name.

```

def OnTextChanged(self, e):

    sb = self.GetStatusBar()
    sb.SetStatusText(' modified', 1)

```

When the text of a text control changes, we write a ' modified' text to the second part of the statusbar. The index to the first part is 0, to the second part 1.

```

def OnOpenFile(self, e):

    if self.te.IsModified():
        dlg = wx.MessageDialog(self, 'Save changes?',
                                '', wx.YES_NO | wx.YES_DEFAULT
                                | wx.CANCEL | wx.ICON_QUESTION)
        ...
    else:
        self.OpenReadFile()

```

The `OnOpenFile` method is called when we select the first menu item. If we want to open a file after we have done some changes, we display a message dialog asking, if we want to save the changes before opening a new file. If there are not any unsaved changes, we just call the `OpenReadFile` method.

```

val = dlg.ShowModal()

if val == wx.ID_YES:
    self.OnSaveFile(e)
    self.OpenReadFile()

elif val == wx.ID_CANCEL:
    dlg.Destroy()

else:
    self.OpenReadFile()

```

The previously mentioned dialog is shown. If we select the Yes button, we save the file and display a dialog for opening a new file in the `OpenReadFile` method. If we select the Cancel button, the dialog is destroyed and nothing is done. If we click on the No button, the `OpenReadFile` method is called and the changes are discarded.

```
def OpenReadFile(self):
    wcd = 'All files (*)|*|Editor files (*.ef)|*.ef'
    ...
```

The `OpenReadFile` method opens the file and reads its contents. In first line of the method, we create a wildcard filter. This filter is used by the dialog window to limit the files shown to specific file patterns. Each filter has two parts: the description and the file pattern. In our case, we have two filters for files which can be chosen: all files or files ending in `.ef` extension.

```
dirname = os.getcwd()

od = wx.FileDialog(self, message='Choose a file',
    defaultDir=dirname, defaultFile='',
    wildcard=wcd, style=wx.FD_OPEN|wx.FD_CHANGE_DIR)
```

The `getcwd` method returns the current working directory. It is needed by the following `wx.FileDialog` constructor. It creates a wxPython standard dialog for selecting one or more files from the filesystem. The `wx.FD_OPEN` flag creates a file dialog for opening a new file and the `wx.FD_CHANGE_DIR` flag changes the current working directory of the dialog to the directory where the file(s) chosen by the user are located.

```
if od.ShowModal() == wx.ID_OK:
    path = od.GetPath()

    fname = None

    try:
        fname = open(path, 'r')
        text = fname.read()

        if self.te.GetLastPosition():
            self.te.Clear()

        self.te.WriteText(text)
        self.last_name_saved = path
        self.te.SetModified(False)

        sb = self.GetStatusBar()
        stat = os.path.basename(path) + ' opened'
        sb.SetStatusText(stat, 0)
        sb.SetStatusText('', 1)
    ...
```

When we click on the OK button of the `wx.FileDialog`, we get the selected file name with the `GetPath` method. We open the file and read its contents. The text control is cleared if there is some text. The contents of the file are written to the text control. The statusbar is updated accordingly.

```
except IOError as e:
    dlg = wx.MessageDialog(self,
        'Error opening file\n' + str(e))
    dlg.ShowModal()
    dlg.Destroy()
```

```
except UnicodeDecodeError as e:

    dlg = wx.MessageDialog(self,
        'Error opening file\n' + str(e))
    dlg.ShowModal()
    dlg.Destroy()
```

We catch for possible IO and Unicode errors.

```
finally:

    if fname:
        fname.close()
```

In the finally block, we close the opened file handler

```
def OnSaveFile(self, e):

    lns = self.last_name_saved

    if lns:
        ...
    else:
        self.OnSaveAsFile(e)
```

The second file menu item calls the `OnSaveFile` method. If there is a file name stored in the `last_name_saved` variable, we save the file. Otherwise, we call the `OnSaveAsFile` method, which opens a dialog for saving a file.

```
try:
    fname = open(lns, 'w')
    text = self.te.GetValue()
    fname.write(text)
```

In this code excerpt, we open a file for writing. We get the contents of the text control and write it to the file.

```
stat = os.path.basename(lns) + ' saved'
sb.SetStatusText(stat, 0)
```

The whole file name may contain a very long path. The `basename` function returns the last part of the file name; this is displayed in the first part of the statusbar.

```
def OnSaveAsFile(self, e):

    wcd='All files(*)|*|Editor files (*.ef)|*.ef'

    directory = os.getcwd()

    sd = wx.FileDialog(self, message='Save file as...',
        defaultDir=directory, defaultFile='',
        wildcard=wcd,
        style=wx.FD_SAVE|wx.FD_OVERWRITE_PROMPT)
    ...
```

In the `OnSaveAsFile` method we show a dialog for saving a file. With the `wx.FD_SAVE` flag we create a file save dialog. The `wx.FD_OVERWRITE_PROMPT` flag

prompts for a confirmation if a file is about to be overwritten.

Chapter 5

wx.ListCtrl

A `wx.ListCtrl` is a graphical representation of a list of items. It is a widget in between a listbox widget and a grid widget. It consists of rows and columns. File managers use list controls to display files and directories on the file system; CD/DVD burners display files to be burned inside a list control.

A `wx.ListCtrl` can be used in three different modes: a report view, list view, or an icon view. These modes are controlled by the `wx.ListCtrl` window styles:

- `wx.LC_REPORT`
- `wx.LC_LIST`
- `wx.LC_ICON`

5.1 Simple example in a report view

In the first example, we create a `wx.ListCtrl` in a report view having three columns. The data from the selected row is shown in the statusbar.

Listing 5.1: A simple `wx.ListCtrl` in a report view

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a simple
wx.ListCtrl in a report view.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sys

class Example(wx.Frame):

    def __init__(self, *args, **kw):
```

```

        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 450))
        self.SetTitle('Simple')
        self.Centre()

def InitUI(self):

    hbox = wx.BoxSizer(wx.HORIZONTAL)

    pnl = wx.Panel(self)

    self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
    self.lc.InsertColumn(0, 'name', width=140)
    self.lc.InsertColumn(1, 'place', width=130)
    self.lc.InsertColumn(2, 'year', width=90)

    hbox.Add(self.lc, 1, wx.EXPAND)
    pnl.SetSizer(hbox)

    self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.OnSelected)
    self.CreateStatusBar()

def InitData(self):

    actresses = [('Jessica Alba', 'Pomona', '1981'),
                 ('Sigourney Weaver', 'New York', '1949'),
                 ('Angelina Jolie', 'Los Angeles', '1975'),
                 ('Natalie Portman', 'Jerusalem', '1981'),
                 ('Rachel Weiss', 'London', '1971'),
                 ('Scarlett Johansson', 'New York', '1984')]

    for i in actresses:
        ix = self.lc.InsertItem(sys.maxsize, i[0])
        self.lc.SetItem(ix, 1, i[1])
        self.lc.SetItem(ix, 2, i[2])

def OnSelected(self, e):

    idx = e.GetIndex()

    txt1 = self.lc.GetItem(idx, 0).GetText()
    txt2 = self.lc.GetItem(idx, 1).GetText()
    txt3 = self.lc.GetItem(idx, 2).GetText()

    text = txt1 + " " + txt2 + " " + txt3
    sb = self.GetStatusBar()
    sb.SetStatusText(text)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

```

```
if __name__ == "__main__":
    main()
```

We have a `wx.ListCtrl` and a statusbar. In the list control, we show six rows of actresses. Each row has three columns.

```
self.InitUI()
self.InitData()
```

In the `InitUI` method, the list control is created. In the `InitData` method, it is filled with data.

```
self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
self.lc.InsertColumn(0, 'name', width=140)
self.lc.InsertColumn(1, 'place', width=130)
self.lc.InsertColumn(2, 'year', width=90)
```

An instance of a `wx.ListCtrl` is created. Three columns are added to the widget using the `InsertColumn` method.

```
self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.OnSelected)
```

When we select a particular row, the `OnSelected` method is called.

```
def InitData(self):

    actresses = [('Jessica Alba', 'Pomona', '1981'),
                  ('Sigourney Weaver', 'New York', '1949'),
                  ('Angelina Jolie', 'Los Angeles', '1975'),
                  ('Natalie Portman', 'Jerusalem', '1981'),
                  ('Rachel Weiss', 'London', '1971'),
                  ('Scarlett Johansson', 'New York', '1984' )]

    for i in actresses:
        ix = self.lc.InsertItem(sys.maxsize, i[0])
        self.lc.SetItem(ix, 1, i[1])
        self.lc.SetItem(ix, 2, i[2])
```

In the `InitData` method, we have a list of actresses. The data is added to the list control. A new row is started with the `InsertItem` method. The first parameter of this method is the row index. If the index is greater than the number of rows in the list control, it is added at the end of it.

Sometimes a `sys.maxsize` value is provided, since we assume there will not be that many rows. The method returns the actual index of the row, which is used by subsequent `SetItem` method calls to add additional items to the row.

```
def OnSelected(self, e):

    idx = e.GetIndex()

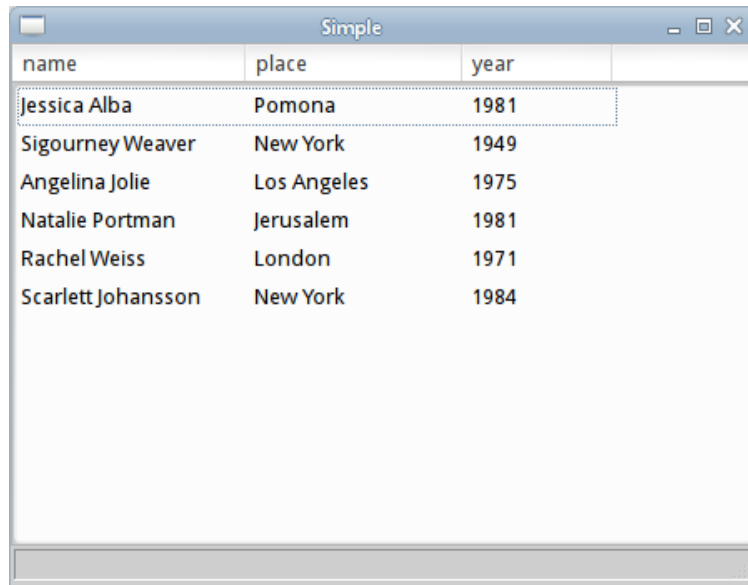
    txt1 = self.lc.GetItem(idx, 0).GetText()
    txt2 = self.lc.GetItem(idx, 1).GetText()
    txt3 = self.lc.GetItem(idx, 2).GetText()
    ...
```

We get the index of the selected row using the `GetIndex` method. We use this

index to retrieve the items of the row with the `GetItem` method.

```
text = txt1 + " " + txt2 + " " + txt3
sb = self.GetStatusBar()
sb.SetStatusText(text)
```

The data is displayed in the statusbar.



name	place	year
Jessica Alba	Pomona	1981
Sigourney Weaver	New York	1949
Angelina Jolie	Los Angeles	1975
Natalie Portman	Jerusalem	1981
Rachel Weiss	London	1971
Scarlett Johansson	New York	1984

Figure 5.1: A `wx.ListCtrl` example in a report view

Figure 5.1 shows a `wx.ListCtrl` having six rows and three columns.

5.2 Selections

It is possible to select one or more rows with a mouse pointer. Multiple rows are selected by clicking on rows and simultaneously holding control or shift keyboard buttons. The background colour of a selected row is changed.

Listing 5.2: Selections

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we work with selections in
ListCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sys
```

```

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 450))
        self.SetTitle('Selections')
        self.Centre()

    def InitUI(self):

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        pnl = wx.Panel(self)

        self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
        self.lc.InsertColumn(0, 'name', width=140)
        self.lc.InsertColumn(1, 'place', width=130)
        self.lc.InsertColumn(2, 'year', width=90)

        nid1 = wx.NewIdRef()
        self.Bind(wx.EVT_MENU, self.OnCtrlQ, id=nid1)

        accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,
            ord('Q'), nid1) ])
        self.SetAcceleratorTable(accel_tbl)

        hbox.Add(self.lc, 1, wx.EXPAND)
        pnl.SetSizer(hbox)

    def InitData(self):

        actresses = [('Jessica Alba', 'Pomona', '1981'),
            ('Sigourney Weaver', 'New York', '1949'),
            ('Angelina Jolie', 'Los Angeles', '1975'),
            ('Natalie Portman', 'Jerusalem', '1981'),
            ('Rachel Weiss', 'London', '1971'),
            ('Scarlett Johansson', 'New York', '1984')]

        for i in actresses:
            idx = self.lc.InsertItem(sys.maxsize,
                i[0])
            self.lc.SetItem(idx, 1, i[1])
            self.lc.SetItem(idx, 2, i[2])

    def OnCtrlQ(self, event):

        indexes = []
        item = -1

        while True:
            idx = self.lc.GetNextSelected(item)

            if idx == -1:

```

```

        break
    else:
        item = idx
        indexes.append(idx)

    print(indexes)

    n_sel = self.lc.GetSelectedItemCount()
    print(f"Number of selected rows: {n_sel}")

    for i in indexes:
        print(self.lc.GetItem(i, 0).GetText())
        print(self.lc.GetItem(i, 1).GetText())
        print(self.lc.GetItem(i, 2).GetText())

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

In the example, we create a Ctrl+Q keyboard shortcut which prints all selected rows to the terminal.

```

nid1 = wx.NewIdRef()
self.Bind(wx.EVT_MENU, self.OnCtrlQ, id=nid1)

accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,
    ord('Q'), nid1)])
self.SetAcceleratorTable(accel_tbl)

```

These lines create a Ctrl+Q keyboard shortcut. The `wx.AcceleratorTable` allows the application to specify a table of keyboard shortcuts for menu or button commands. A new Id for the keyboard shortcut is generated with `wx.NewIdRef`.

```

indexes = []
item = -1

while True:
    idx = self.lc.GetNextSelected(item)

    if idx == -1:
        break
    else:
        item = idx
        indexes.append(idx)

print(indexes)

```

Using the `GetNextSelected` method, we gather all the indexes of rows that are being selected. If the method takes -1 as a parameter, it returns the first selected row.

```
n_sel = self.lc.GetSelectedItemCount()
print(f"Number of selected rows: {n_sel}")
```

The `GetSelectedItemCount` method returns the number of selected rows in the list control.

```
for i in indexes:
    print(self.lc.GetItem(i, 0).GetText())
    print(self.lc.GetItem(i, 1).GetText())
    print(self.lc.GetItem(i, 2).GetText())
```

In the for loop, we print all the data from the selected rows.

5.3 Editable list control

In the two previous examples, we could not edit the data in the list control. The data can be modified using a specialized text edit mixin.

Listing 5.3: Editable `wx.ListCtrl`

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create an editable
wx.ListCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sys
import wx.lib.mixins.listctrl as lm

class EditableListCtrl(wx.ListCtrl, lm.TextEditMixin):

    def __init__(self, *args, **kw):

        super(EditableListCtrl, self).__init__(*args, **kw)
        lm.TextEditMixin.__init__(self)

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 450))
        self.SetTitle('Editable')
        self.Centre()

    def InitUI(self):
```

```

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        pnl = wx.Panel(self)

        self.lc = EditableListCtrl(pnl, style=wx.LC_REPORT)
        self.lc.InsertColumn(0, 'name', width=140)
        self.lc.InsertColumn(1, 'place', width=130)
        self.lc.InsertColumn(2, 'year', width=90)

        hbox.Add(self.lc, 1, wx.EXPAND)
        pnl.SetSizer(hbox)

    def InitData(self):

        actresses = [('Jessica Alba', 'Pomona', '1981'),
                     ('Sigourney Weaver', 'New York', '1949'),
                     ('Angelina Jolie', 'Los angeles', '1975'),
                     ('Natalie Portman', 'Jerusalem', '1981'),
                     ('Rachel Weiss', 'London', '1971'),
                     ('Scarlett Johansson', 'New York', '1984')]

        for i in actresses:

            idx = self.lc.InsertItem(sys.maxsize, i[0])
            self.lc.SetItem(idx, 1, i[1])
            self.lc.SetItem(idx, 2, i[2])

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

This example shows how to create an editable `wx.ListCtrl`.

```
import wx.lib.mixins.listctrl as lm
```

We import the module that contains the appropriate mixin.

```

class EditableListCtrl(wx.ListCtrl, lm.TextEditMixin):

    def __init__(self, *args, **kw):

        super(EditableListCtrl, self).__init__(*args, **kw)
        lm.TextEditMixin.__init__(self)

```

We create our custom list control class. It inherits both from the `wx.ListCtrl` and from the `lm.TextEditMixin`.

```
self.lc = EditableListCtrl(pnl, style=wx.LC_REPORT)
```

An instance of our custom derived list control is created.

5.4 Background color & auto width mixin

The fourth example dedicated to the `wx.ListCtrl` will change the background colour of the rows. In addition, we will use an auto width mixin. This mixin automatically resizes the last column to take up the remaining width of the list control.

Listing 5.4: Background color and autowidth mixin

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we change the colour of every second
row and use an autowidth mixin.
"""

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sys
from wx.lib.mixins.listctrl import ListCtrlAutoWidthMixin

class AutoWidthListCtrl(wx.ListCtrl, ListCtrlAutoWidthMixin):

    def __init__(self, *args, **kw):
        super(AutoWidthListCtrl, self).__init__(*args, **kw)
        ListCtrlAutoWidthMixin.__init__(self)

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 450))
        self.SetTitle('Background colour & autowidth')
        self.Centre()

    def InitUI(self):

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        pnl = wx.Panel(self)

        self.lc = AutoWidthListCtrl(pnl, style=wx.LC_REPORT)
        self.lc.InsertColumn(0, 'name', width=140)
        self.lc.InsertColumn(1, 'place', width=130)
        self.lc.InsertColumn(2, 'year', width=90)

        hbox.Add(self.lc, 1, wx.EXPAND)
        pnl.SetSizer(hbox)
```

```

def InitData(self):

    actresses = [('Jessica Alba', 'Pomona', '1981'),
                  ('Sigourney Weaver', 'New York', '1949'),
                  ('Angelina Jolie', 'Los Angeles', '1975'),
                  ('Natalie Portman', 'Jerusalem', '1981'),
                  ('Rachel Weiss', 'London', '1971'),
                  ('Scarlett Johansson', 'New York', '1984')]

    for i in actresses:

        idx = self.lc.InsertItem(sys.maxsize, i[0])
        self.lc.SetItem(idx, 1, i[1])
        self.lc.SetItem(idx, 2, i[2])

        if idx % 2:
            self.lc.SetItemBackgroundColour(idx, "#3a3a3a")

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

Every second row in the list control has a gray background colour. Also an auto width mixin is used.

```
from wx.lib.mixins.listctrl import ListCtrlAutoWidthMixin
```

We import the `ListCtrlAutoWidthMixin` from the `wx.lib.mixins.listctrl` module.

```
class AutoWidthListCtrl(wx.ListCtrl, ListCtrlAutoWidthMixin):
    ...
```

The mixin is inherited by our custom class.

```
if idx % 2:
    self.lc.SetItemBackgroundColour(idx, "gray")
```

The background colour of every second row is changed to gray colour using the `SetItemBackgroundColour` method.

name	place	year
Jessica Alba	Pomona	1981
Sigourney Weaver	New York	1949
Angelina Jolie	Los Angeles	1975
Natalie Portman	Jerusalem	1981
Rachel Weiss	London	1971
Scarlett Johansson	New York	1984

Figure 5.2: Background colour and auto width mixin

Figure 5.2 shows every second row with a gray background color. We can also see the third column taking all the remaining space.

5.5 Images in a report view

Images can be inserted into the `wx.ListCtrl` in a report view. They are shown before the first column, in the very left of the list control.

Listing 5.5: Images in a report view

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we add images to the
list control in a report view.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sys

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
```

```

        self.InitData()

        self.SetSize((750, 450))
        self.SetTitle('Images')
        self.Centre()

    def InitUI(self):

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        pnl = wx.Panel(self)

        self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
        self.lc.InsertColumn(0, 'Name', width=170)
        self.lc.InsertColumn(1, 'Ext', width=100)

        hbox.Add(self.lc, 1, wx.EXPAND)
        pnl.SetSizer(hbox)

    def InitData(self):

        fls = [('iconview.py', 'py'), ('images', ''),
               ('advanced_wxpython', 'pdf'), ('notes', 'txt')]

        imgs = wx.ImageList(16, 16)
        imgs.Add(wx.Bitmap('python.png'))
        imgs.Add(wx.Bitmap('folder.png'))
        imgs.Add(wx.Bitmap('pdf.png'))
        imgs.Add(wx.Bitmap('file.png'))

        self.lc.AssignImageList(imgs, wx.IMAGE_LIST_SMALL)

        for i in fls:
            ix = self.lc.InsertItem(sys.maxsize, i[0])
            self.lc.SetItem(ix, 1, i[1])

        for i in range(4):
            self.lc.SetItemImage(i, i)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

The example shows a `wx.ListCtrl` with two columns. They show a file name and its extension. The images assigned to the list control represent the file type.

```

self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
self.lc.InsertColumn(0, 'Name', width=170)
self.lc.InsertColumn(1, 'Ext', width=100)

```

A `wx.ListCtrl` is created in a report view. Two columns are inserted.

```
fls = [('iconview.py', 'py'), ('images', ''),
      ('advanced_wxpython', 'pdf'), ('notes', 'txt')]
```

These are file names and their extensions to be shown in the list control.

```
imgs = wx.ImageList(16, 16)
imgs.Add(wx.Bitmap('python.png'))
imgs.Add(wx.Bitmap('folder.png'))
imgs.Add(wx.Bitmap('pdf.png'))
imgs.Add(wx.Bitmap('file.png'))
```

An image list consisting of four images is created. Each of the images is 16x16.

```
self.lc.AssignImageList(imgs, wx.IMAGE_LIST_SMALL)
```

Images are assigned to the list control.

```
for i in fls:
    ix = self.lc.InsertItem(sys.maxsize, i[0])
    self.lc.SetItem(ix, 1, i[1])
```

The text data is inserted into the list control.

```
for i in range(4):
    self.lc.SetItemImage(i, i)
```

The images are inserted into the list control using the `SetItemImage` method. The first parameter is the row index, the second one is the image index in the image list.

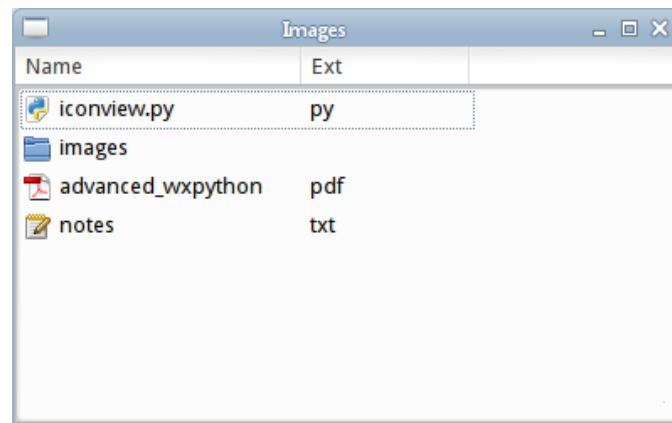


Figure 5.3: Images in a report view

Figure 5.3 shows a `wx.ListCtrl` widget in a report view with images.

5.6 Icon view

Next we create a `wx.ListCtrl` in an icon view. In this mode the items are represented by images and optional labels. An icon view is created with the `wx.LC_ICON` window style flag.

Listing 5.6: Icon view

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a wx.ListCtrl
in an icon view.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((450, 150))
        self.SetTitle('Icon view')
        self.Centre()

    def InitUI(self):

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        pnl = wx.Panel(self)

        self.lc = wx.ListCtrl(pnl, style=wx.LC_ICON)

        hbox.Add(self.lc, 1, wx.EXPAND)
        pnl.SetSizer(hbox)

    def InitData(self):

        self.imgs = wx.ImageList(64, 64)

        self.imgs.Add(wx.Bitmap('firefox.png'))
        self.imgs.Add(wx.Bitmap('firefox.png'))
        self.imgs.Add(wx.Bitmap('firefox.png'))
        self.imgs.Add(wx.Bitmap('firefox.png'))
        self.imgs.Add(wx.Bitmap('firefox.png'))

        self.lc.AssignImageList(self.imgs,
                                wx.IMAGE_LIST_NORMAL)
        self.lc.InsertItem(0, 0)
        self.lc.InsertItem(1, 1)
        self.lc.InsertItem(2, 2)
        self.lc.InsertItem(3, 3)
        self.lc.InsertItem(4, 4)
```

```
def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()
```

In the example we show five images in the `wx.ListCtrl`.

```
self.lc = wx.ListCtrl(pnl, style=wx.LC_ICON)
```

An instance of the `wx.ListCtrl` is created. The `wx.LC_ICON` shows it in the icon view mode.

```
self.imgs = wx.ImageList(64, 64)

self.imgs.Add(wx.Bitmap('firefox.png'))
self.imgs.Add(wx.Bitmap('firefox.png'))
self.imgs.Add(wx.Bitmap('firefox.png'))
self.imgs.Add(wx.Bitmap('firefox.png'))
self.imgs.Add(wx.Bitmap('firefox.png'))
```

An image list containing five bitmaps is created. Each bitmap is 64x64.

```
self.lc.AssignImageList(self.imgs,
    wx.IMAGE_LIST_NORMAL)
```

The `AssignImageList` method assigns images to the list control.

```
self.lc.InsertImageItem(0, 0)
self.lc.InsertImageItem(1, 1)
self.lc.InsertImageItem(2, 2)
self.lc.InsertImageItem(3, 3)
self.lc.InsertImageItem(4, 4)
```

The `InsertImageItem` method inserts an image item to the list control. All the five images are inserted. The first parameter is the position in the list control, the second one is the image index from the image list.



Figure 5.4: Icon view of the `wx.ListCtrl`

Figure 5.4 shows the `wx.ListCtrl` in the icon view mode.

5.7 List control with check boxes

The `EnableCheckBoxes` method enables or disables checkboxes for list items.

Listing 5.7: `wx.ListCtrl` with check boxes

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we show a check box in
a list control.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import sys

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 350))
        self.SetTitle('Check list')
        self.Centre()

    def InitUI(self):

        pnl = wx.Panel(self)

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        lpnl = wx.Panel(pnl)
        rpnl = wx.Panel(pnl)

        self.log = wx.TextCtrl(rpnl, style=wx.TE_MULTILINE)

        self.lc = wx.ListCtrl(rpnl, style=wx.LC_REPORT)
        self.lc.EnableCheckBoxes()
        self.lc.InsertColumn(0, 'name', width=140)
        self.lc.InsertColumn(1, 'place', width=130)
        self.lc.InsertColumn(2, 'year', width=90)

        vbox2 = wx.BoxSizer(wx.VERTICAL)

        sel = wx.Button(lpnl, label='Select All',
                        size=(100, -1))
        des = wx.Button(lpnl, label='Deselect All',
                        size=(100, -1))
        app = wx.Button(lpnl, label='Select',
                        size=(100, -1))
```



```

sel.Bind(wx.EVT_BUTTON, self.OnSelectAll)
des.Bind(wx.EVT_BUTTON, self.OnDeselectAll)
app.Bind(wx.EVT_BUTTON, self.OnSelect)

vbox2.Add(sel, 0, wx.TOP, 5)
vbox2.Add(des)
vbox2.Add(app)

lpnl.SetSizer(vbox2)

vbox.Add(self.lc, 1, wx.EXPAND | wx.TOP, 3)
vbox.Add((-1, 10))
vbox.Add(self.log, 1, wx.EXPAND)
vbox.Add((-1, 10))

rpnl.SetSizer(vbox)

hbox.Add(lpnl, 0, wx.EXPAND | wx.RIGHT, 5)
hbox.Add(rpnl, 1, wx.EXPAND)
hbox.Add((3, -1))

pnl.SetSizer(hbox)

def InitData(self):

    actresses = [('Jessica Alba', 'Pomona', '1981'),
                  ('Sigourney Weaver', 'New York', '1949'),
                  ('Angelina Jolie', 'Los Angeles', '1975'),
                  ('Natalie Portman', 'Jerusalem', '1981'),
                  ('Rachel Weiss', 'London', '1971'),
                  ('Scarlett Johansson', 'New York', '1984' )]

    for i in actresses:
        ix = self.lc.InsertItem(sys.maxsize, i[0])
        self.lc.SetItem(ix, 1, i[1])
        self.lc.SetItem(ix, 2, i[2])

def OnSelectAll(self, e):

    num = self.lc.GetItemCount()
    for i in range(num):
        self.lc.CheckItem(i)

def OnDeselectAll(self, e):

    num = self.lc.GetItemCount()
    for i in range(num):
        self.lc.CheckItem(i, False)

def OnSelect(self, e):

    num = self.lc.GetItemCount()

    for i in range(num):

        if i == 0:
            self.log.Clear()

```

```

        if self.lc.IsSelected(i):
            txt = self.lc.GetItemText(i) + '\n'
            self.log.AppendText(txt)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

The example has three buttons which select or deselect rows in the list control.

```
self.log = wx.TextCtrl(rpnl, style=wx.TE_MULTILINE)
```

Items selected with the third button are shown in a `wx.TextCtrl`.

```

sel = wx.Button(lpnl, label='Select All',
               size=(100, -1))
des = wx.Button(lpnl, label='Deselect All',
               size=(100, -1))
app = wx.Button(lpnl, label='Select',
               size=(100, -1))

```

We have three buttons in the left part of the window. The first checks all check boxes in the list control. The second unchecks all boxes. The third button writes the names of the selected actresses into the log window.

```

def OnSelectAll(self, e):

    num = self.lc.GetItemCount()
    for i in range(num):
        self.lc.CheckItem(i)

```

This method checks all the check boxes in the list control. We determine the number of rows in the list control with `GetItemCount`. We check the boxes with the `CheckItem` method.

```

def OnSelect(self, e):

    num = self.lc.GetItemCount()

    for i in range(num):

        if i == 0:
            self.log.Clear()

        if self.lc.IsChecked(i):
            txt = self.lc.GetItemText(i) + '\n'
            self.log.AppendText(txt)

```

The names of the actresses of the rows with checked boxes are written to the log window.

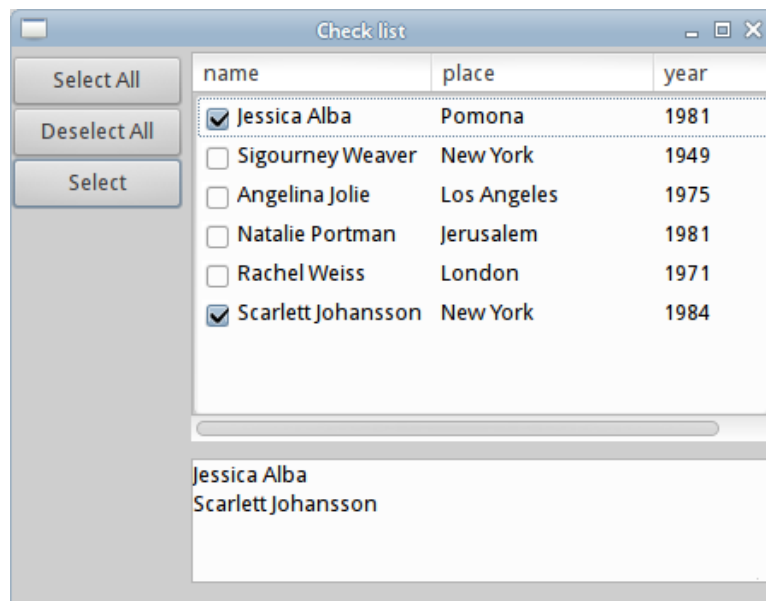


Figure 5.5: wx.ListCtrl with check boxes

Figure 5.5 shows a wx.ListCtrl with the CheckListCtrlMixin.

5.8 Sortable columns

Next we show how to create a list control with sortable columns. There is a ColumnSorterMixin for this task. It handles sorting of columns in a report view mode. There are several requirements for this mixin. There must be a GetListCtrl method which returns the list control object. Items in the list control must have a unique data value set with the SetItemData method. The ColumnSorterMixin.__init__ method must be called after the list control object was instantiated. Finally, there must be an attribute called itemDataMap which holds the dictionary of the data which is shown in the list control.

The following example works on Linux. On Windows, the underlying native control behaves differently when we assign an image list to it. Windows users will find a specific example for Windows. The example is called `sorting_win.py`. The explanations are provided in the comments of the example.

Listing 5.8: Sorting columns

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a wx.ListCtrl with
sortable columns.

Author: Jan Bodnar
Website: zetcode.com
"""
```

```

import wx
import sys
from wx.lib.mixins.listctrl import ColumnSorterMixin

class Example(wx.Frame, ColumnSorterMixin):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

        self.SetSize((750, 450))
        self.SetTitle('Sorting')
        self.Centre()

    def InitUI(self):

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        pnl = wx.Panel(self)

        self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
        self.lc.InsertColumn(0, 'name', width=140)
        self.lc.InsertColumn(1, 'place', width=130)
        self.lc.InsertColumn(2, 'year', width=90)

        self.InitData()

        actresses = {
            1 : ('Jessica Alba', 'Pomona', '1981'),
            2 : ('Sigourney Weaver', 'New York', '1949'),
            3 : ('Angelina Jolie', 'Los Angeles', '1975'),
            4 : ('Natalie Portman', 'Jerusalem', '1981'),
            5 : ('Rachel Weiss', 'London', '1971'),
            6 : ('Scarlett Johansson', 'New York', '1984')
        }

        self.itemDataMap = actresses
        ColumnSorterMixin.__init__(self, 3)

        hbox.Add(self.lc, 1, wx.EXPAND)
        pnl.SetSizer(hbox)

        self.imgs = wx.ImageList(16, 16)
        imup = wx.ArtProvider.GetBitmap(wx.ART_GO_UP,
            wx.ART_TOOLBAR, (16, 16))
        self.sdown = self.imgs.Add(imup)

        imdown = wx.ArtProvider.GetBitmap(wx.ART_GO_DOWN,
            wx.ART_TOOLBAR, (16, 16))
        self.sup = self.imgs.Add(imdown)

        self.lc.SetImageList(self.imgs, wx.IMAGE_LIST_SMALL)

    def GetSortImages(self):
        return (self.sdown, self.sup)

```

```

def GetListCtrl(self):
    return self.lc

def InitData(self):

    actresses = {
        1 : ('Jessica Alba', 'Pomona', '1981'),
        2 : ('Sigourney Weaver', 'New York', '1949'),
        3 : ('Angelina Jolie', 'Los Angeles', '1975'),
        4 : ('Natalie Portman', 'Jerusalem', '1981'),
        5 : ('Rachel Weiss', 'London', '1971'),
        6 : ('Scarlett Johansson', 'New York', '1984')
    }

    for key, data in actresses.items():

        ix = self.lc.InsertItem(sys.maxsize, data[0])
        self.lc.SetItem(ix, 1, data[1])
        self.lc.SetItem(ix, 2, data[2])
        self.lc.SetItemData(ix, key)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

In this example, the columns are sorted if we click on them. There is also an arrow indicating whether the sorting is in the ascending or descending mode.

```
from wx.lib.mixins.listctrl import ColumnSorterMixin
```

This is the required mixin for sortable columns.

```
class Example(wx.Frame, ColumnSorterMixin):
    ...
```

The `ColumnSorterMixin` is not used in a separate custom list control class. We use it in the `Example` class. The mixin must be initiated after the list control was created.

```

self.lc = wx.ListCtrl(pnl, style=wx.LC_REPORT)
self.lc.InsertColumn(0, 'name', width=140)
self.lc.InsertColumn(1, 'place', width=130)
self.lc.InsertColumn(2, 'year', width=90)

```

A `wx.ListCtrl` is created with three columns.

```

for key, data in actresses.items():

    ix = self.lc.InsertItem(sys.maxsize, data[0])

```

```

self.lc.SetItem(ix, 1, data[1])
self.lc.SetItem(ix, 2, data[2])
self.lc.SetItemData(ix, key)

```

While populating the list control, we use the `SetItemData` method to add a unique number for each row in the list control.

```

self.itemDataMap = actresses

```

The `itemDataMap` attribute holds all dictionary data.

```

ColumnSorterMixin.__init__(self, 3)

```

The `ColumnSorterMixin` is initiated. The parameter is the number of columns in the list control. The mixin uses the `wx.ListCtrl!GetListCtrl` method to get the list control object.

```

self.imgs = wx.ImageList(16, 16)
imup = wx.ArtProvider.GetBitmap(wx.ART_GO_UP,
    wx.ART_TOOLBAR, (16, 16))
self.sdown = self.imgs.Add(imup)

imdown = wx.ArtProvider.GetBitmap(wx.ART_GO_DOWN,
    wx.ART_TOOLBAR, (16, 16))
self.sup = self.imgs.Add(imdown)

self.lc.SetImageList(self.imgs, wx.IMAGE_LIST_SMALL)

```

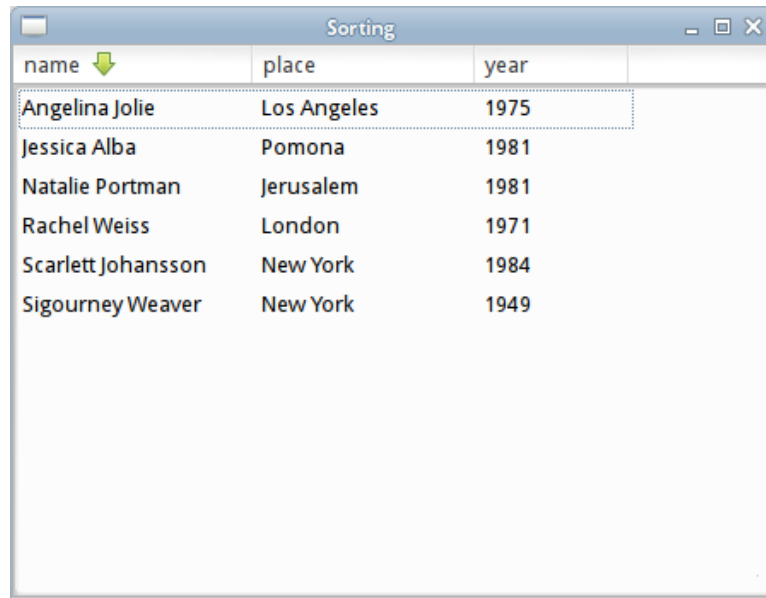
We use two images from the art provider to indicate the sorting mode of a column. We use up and down arrow images.

```

def GetSortImages(self):
    return (self.sdown, self.sup)

```

This method is used by the mixin to get the images that are displayed in the column.



name	place	year
Angelina Jolie	Los Angeles	1975
Jessica Alba	Pomona	1981
Natalie Portman	Jerusalem	1981
Rachel Weiss	London	1971
Scarlett Johansson	New York	1984
Sigourney Weaver	New York	1949

Figure 5.6: Sortable columns

Figure 5.6 shows a `wx.ListCtrl` with the first column sorted in an ascending mode.

5.9 Virtual list control

There are applications which need to handle large amounts of data. This could be millions of data items. A lot of memory would be consumed in such cases. Therefore; wxPython has a special virtual mode of the `wx.ListCtrl`.

In this mode, the data is loaded only when it is needed to be displayed on the screen. A virtual list control is created with the `wx.LC_VIRTUAL` flag. We must also call the `SetItemCount` method to indicate the number of rows for the list control.

We must also implement at least the `OnGetItemText` method which returns the string containing the text for the given column/row. The `OnGetItemImage`, `OnGetItemColumnImage` and `OnGetItemAttr` methods must be implemented if we work with images or modify the appearance of the list items.

Listing 5.9: Virtual list control

```
#!/usr/bin/env python
'''
ZetCode Advanced wxPython tutorial

In this program we create a virtual
wx.ListCtrl.

Author: Jan Bodnar
Website: zetcode.com
'''
```

```

import wx

class VirtualListCtrl(wx.ListCtrl):

    def __init__(self, *args, **kw):

        super(VirtualListCtrl, self).__init__(*args,
            style=wx.LC_REPORT | wx.LC_VIRTUAL)

        self.InitUI()
        self.InitData()

    def InitUI(self):

        self.InsertColumn(0, 'Column 1', width=140)
        self.InsertColumn(1, 'Column 2', width=140)
        self.InsertColumn(2, 'Column 3', width=140)

    def InitData(self):

        n = 10000
        self.data = []

        for i in range(n):

            i1 = "Row %d, Column 1" % i
            i2 = "Row %d, Column 2" % i
            i3 = "Row %d, Column 3" % i

            self.data.append((i1, i2, i3))

        self.SetItemCount(n)

    def OnGetItemText(self, item, col):

        return self.data[item][col]

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

        self.SetSize((750, 450))
        self.SetTitle('Virtual list control')
        self.Centre()

    def InitUI(self):

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        pnl = wx.Panel(self)
        lc = VirtualListCtrl(pnl)

        hbox.Add(lc, 1, wx.EXPAND)

```



```

        pnl.SetSizer(hbox)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

In the example, we create a virtual list control which contains three columns and ten thousand rows.

```

class VirtualListCtrl(wx.ListCtrl):

    def __init__(self, *args, **kw):

        super(VirtualListCtrl, self).__init__(*args,
            style=wx.LC_REPORT|wx.LC_VIRTUAL)

        self.InitUI()
        self.InitData()

```

The `wx.LC_VIRTUAL` is a flag to create a virtual list control.

```

def InitData(self):

    n = 10000
    self.data = []

    for i in range(n):

        i1 = "Row %d, Column 1" % i
        i2 = "Row %d, Column 2" % i
        i3 = "Row %d, Column 3" % i

        self.data.append((i1, i2, i3))
    ...

```

In the `InitData` method, we create the data source. It is a list of 10000 tuples. Each tuple has three strings.

```

self.SetItemCount(n)

```

One of the requirements of the virtual list control is to call the `SetItemCount` method. It indicates the number of items (rows) in the list control.

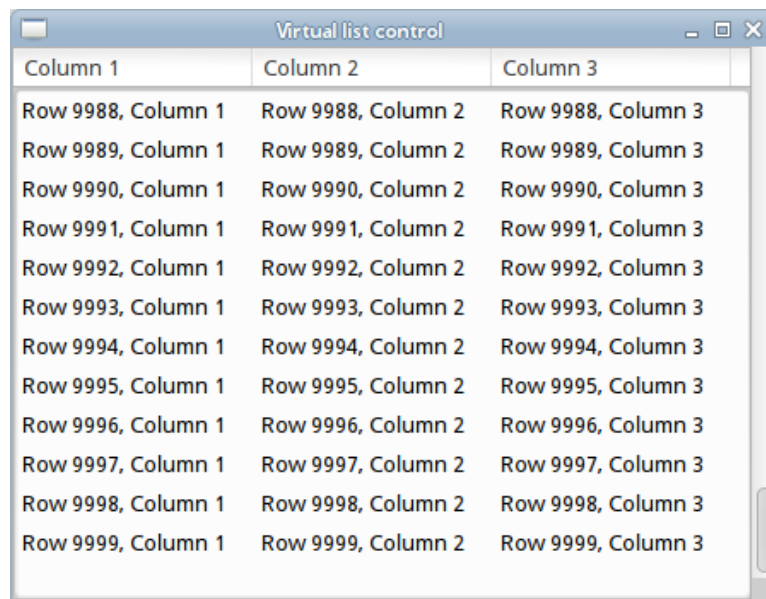
```

def OnGetItemText(self, item, col):

    return self.data[item][col]

```

The `OnGetItemText` method returns the string for the given row/column. We return the text from our data source. The method is called by the virtual list control when needed.



Column 1	Column 2	Column 3
Row 9988, Column 1	Row 9988, Column 2	Row 9988, Column 3
Row 9989, Column 1	Row 9989, Column 2	Row 9989, Column 3
Row 9990, Column 1	Row 9990, Column 2	Row 9990, Column 3
Row 9991, Column 1	Row 9991, Column 2	Row 9991, Column 3
Row 9992, Column 1	Row 9992, Column 2	Row 9992, Column 3
Row 9993, Column 1	Row 9993, Column 2	Row 9993, Column 3
Row 9994, Column 1	Row 9994, Column 2	Row 9994, Column 3
Row 9995, Column 1	Row 9995, Column 2	Row 9995, Column 3
Row 9996, Column 1	Row 9996, Column 2	Row 9996, Column 3
Row 9997, Column 1	Row 9997, Column 2	Row 9997, Column 3
Row 9998, Column 1	Row 9998, Column 2	Row 9998, Column 3
Row 9999, Column 1	Row 9999, Column 2	Row 9999, Column 3

Figure 5.7: Virtual list control

Figure 5.7 shows a virtual list control having three columns and ten thousand rows.

Chapter 6

wx.TreeCtrl

A `wx.TreeCtrl` is a widget designed for working with hierarchical data. The data is displayed in a tree where branches of the tree may be expanded to show further items. The starting point is called a root, which may be hidden if needed.

The `wx.TR_DEFAULT_STYLE` is the default window style for the tree control. It is a combination of flags that are closest to the defaults for the native control.

6.1 Simple example

In the first example we create a simple `wx.TreeCtrl`. It shows some basic functionality of the widget.

Listing 6.1: Simple example

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create a simple
wx.TreeCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 400))
        self.SetTitle('Simple')
        self.Centre()
```

```

def InitUI(self):

    self.tree = wx.TreeCtrl(self)

    self.tree.Bind(wx.EVT_TREE_SEL_CHANGED,
                   self.OnSelChanged)

def InitData(self):

    root = self.tree.AddRoot('Cities')

    self.tree.AppendItem(root, 'New York')
    self.tree.AppendItem(root, 'Bratislava')
    self.tree.AppendItem(root, 'Budapest')
    self.tree.AppendItem(root, 'London')
    self.tree.AppendItem(root, 'Prague')
    self.tree.AppendItem(root, 'Wien')
    self.tree.AppendItem(root, 'Oslo')
    self.tree.AppendItem(root, 'Berlin')

def OnSelChanged(self, e):

    item = e.GetItem()
    print(self.tree.GetItemText(item))

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We have a tree of cities. The selected item is printed to the console.

```

def InitUI(self):

    self.tree = wx.TreeCtrl(self)

    self.tree.Bind(wx.EVT_TREE_SEL_CHANGED,
                   self.OnSelChanged)

```

A `wx.TreeCtrl` is created. The `wx.EVT_TREE_SEL_CHANGED` event is triggered whenever a new item in the tree control is selected.

```

root = self.tree.AddRoot('Cities')

```

A root node is added to the tree control using the `AddRoot` method. It is a starting point of the widget.

```

self.tree.AppendItem(root, 'New York')
self.tree.AppendItem(root, 'Bratislava')
...

```

We add items to the root node with the `AppendItem` method.

```
def OnSelChanged(self, e):  
    item = e.GetItem()  
    print(self.tree.GetItemText(item))
```

The `OnSelChanged` method is called when a new item is selected. We get the tree item with `GetItemText`. The text is printed to the console.

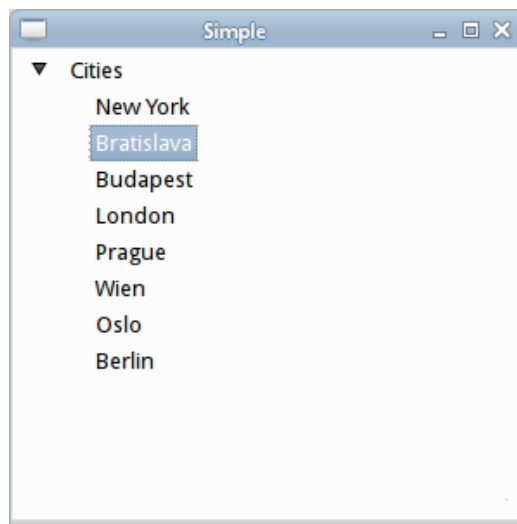


Figure 6.1: A simple example of the `wx.TreeCtrl`

Figure 6.1 shows a few cities in a `wx.TreeCtrl` widget.

6.2 Selections

One or more items of a `wx.TreeCtrl` can be selected by a user. The items can be selected with the mouse pointer or with keyboard buttons. Multiple items in a `wx.TreeCtrl` can be selected by using control and shift keyboard buttons. The background colour of a selected element will change.

Listing 6.2: Selections

```
#!/usr/bin/env python  
  
"""  
ZetCode Advanced wxPython tutorial  
  
In this program we work with  
selections in wx.TreeCtrl.  
  
Author: Jan Bodnar  
Website: zetcode.com  
"""  
  
import wx
```

```

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 450))
        self.SetTitle('Selections')
        self.Centre()

    def InitUI(self):

        pnl = wx.Panel(self)

        self.tree = wx.TreeCtrl(pnl,
                                style=wx.TR_HAS_BUTTONS|wx.TR_LINES_AT_ROOT|
                                    wx.TR_MULTIPLE)

        vbox = wx.BoxSizer(wx.VERTICAL)
        vbox.Add(self.tree, 1, flag=wx.EXPAND)

        nid1 = wx.NewIdRef()
        self.Bind(wx.EVT_MENU, self.OnCtrlL, id=nid1)

        accel_tbl = wx.AcceleratorTable(
            [(wx.ACCEL_CTRL, ord('L'), nid1)])
        self.SetAcceleratorTable(accel_tbl)

        self.log = wx.TextCtrl(pnl, style=wx.TE_MULTILINE)
        vbox.Add(self.log, proportion=1,
                 flag=wx.EXPAND|wx.TOP, border=10)

        pnl.SetSizer(vbox)

    def InitData(self):

        root = self.tree.AddRoot('Cities')

        self.tree.AppendItem(root, 'New York')
        self.tree.AppendItem(root, 'Bratislava')
        self.tree.AppendItem(root, 'Budapest')
        self.tree.AppendItem(root, 'London')
        self.tree.AppendItem(root, 'Prague')
        self.tree.AppendItem(root, 'Wien')
        self.tree.AppendItem(root, 'Oslo')
        self.tree.AppendItem(root, 'Berlin')

    def OnCtrlL(self, e):

        if not self.log.IsEmpty():
            self.log.Clear()

        sels = self.tree.GetSelections()

        for s in sels:
            txt = self.tree.GetItemText(s) + '\n'

```

```

        self.log.AppendText(txt)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

Multiple items of a tree can be selected in this example. With the Ctrl+L keyboard shortcut, the selected items are shown in the log window.

```

self.tree = wx.TreeCtrl(pnl,
    style=wx.TR_HAS_BUTTONS|wx.TR_LINES_AT_ROOT|
    wx.TR_MULTIPLE)

```

The `wx.TR_MULTIPLE` flag allows multiple items of a tree to be selected. The `wx.TR_HAS_BUTTONS` and the `wx.TR_LINES_AT_ROOT` are flags needed for displaying expand buttons on the tree widget on Windows. On Linux, the `wx.TR_LINES_AT_ROOT` flag is not needed.

```

nid1 = wx.NewIdRef()
self.Bind(wx.EVT_MENU, self.OnCtrlL, id=nid1)

accel_tbl = wx.AcceleratorTable(
    [(wx.ACCEL_CTRL, ord('L'), nid1)])
self.SetAcceleratorTable(accel_tbl)

```

A Ctrl+L keyboard shortcut is created. It calls the `OnCtrlL` method.

```

self.log = wx.TextCtrl(pnl, style=wx.TE_MULTILINE)
vbox.Add(self.log, proportion=0.3,
    flag=wx.EXPAND|wx.TOP, border=10)

```

In this text control we show the selected tree items.

```

def OnCtrlL(self, e):

    if not self.log.IsEmpty():
        self.log.Clear()
    ...

```

If the log window is not empty, we clear it.

```

sels = self.tree.GetSelections()

for s in sels:
    txt = self.tree.GetItemText(s) + '\n'
    self.log.AppendText(txt)

```

The `GetSelections` method returns a list of selected items. It is a list of `wx.TreeItemId` objects. The `wx.TreeItemId` is used by the `GetItemText` to get the text of the current element. The text is appended to the log window.

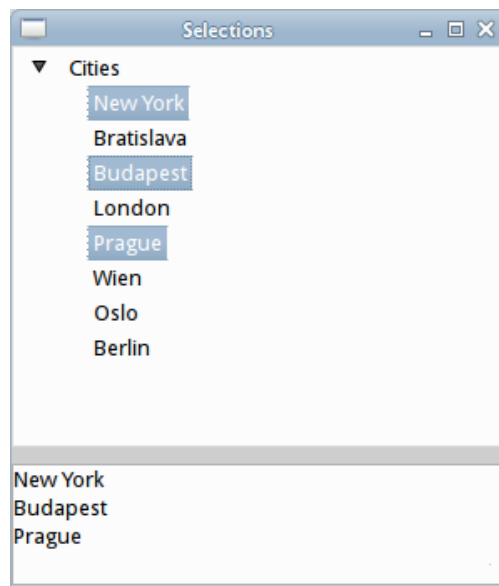


Figure 6.2: wx.TreeCtrl selections

Figure 6.2 shows multiple items selected in a `wx.TreeCtrl`.

6.3 Traversing a tree control

The following example shows how to go through all items of a `wx.TreeCtrl`.

Listing 6.3: Traversing

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we traverse all items
of a wx.TreeCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 400))
        self.SetTitle('Traverse')
        self.Centre()
```



```

def InitUI(self):

    self.tree = wx.TreeCtrl(self)

    vbox = wx.BoxSizer(wx.VERTICAL)
    vbox.Add(self.tree, proportion=1, flag=wx.EXPAND)

    nid = wx.NewIdRef()

    self.Bind(wx.EVT_MENU, self.OnCtrlT, id=nid)

    accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,
        ord('T'), nid)])
    self.SetAcceleratorTable(accel_tbl)

    self.SetSizer(vbox)

def InitData(self):

    root = self.tree.AddRoot('Cities')

    us = self.tree.AppendItem(root, 'United States')
    eu = self.tree.AppendItem(root, 'Europe')
    aus = self.tree.AppendItem(root, 'Australia')

    mas = self.tree.AppendItem(us, 'Massachusetts')
    swe = self.tree.AppendItem(eu, 'Sweden')
    pol = self.tree.AppendItem(eu, 'Poland')
    vic = self.tree.AppendItem(aus, 'Victoria')

    self.tree.AppendItem(mas, 'Gardner')
    stc = self.tree.AppendItem(swe, 'Stockholm')
    self.tree.AppendItem(swe, 'Uppsala')
    self.tree.AppendItem(swe, 'Karlstad')
    self.tree.AppendItem(pol, 'Warsaw')
    self.tree.AppendItem(pol, 'Krakow')
    self.tree.AppendItem(vic, 'Morwell')
    self.tree.AppendItem(vic, 'Melbourne')

    self.tree.AppendItem(stc, 'Population: 847000')

def OnCtrlT(self, e):

    rootItem = self.tree.GetRootItem()
    item, _ = self.tree.GetFirstChild(rootItem)
    self.Traverse(item)

def Traverse(self, item):

    while item.IsOk():

        print(self.tree.GetItemText(item))
        child, _ = self.tree.GetFirstChild(item)

        if child.IsOk():
            self.Traverse(child)

```

```

        item = self.tree.GetNextSibling(item)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

There are multiple branches of a tree control. With the Ctrl+T keyboard shortcut, we go through all items of a tree control and print them to the console.

```

us = self.tree.AppendItem(root, 'United States')
eu = self.tree.AppendItem(root, 'Europe')
aus = self.tree.AppendItem(root, 'Australia')

```

Three items are added to the root item using the `AppendItem` method.

```

mas = self.tree.AppendItem(us, 'Massachusetts')
swe = self.tree.AppendItem(eu, 'Sweden')
pol = self.tree.AppendItem(eu, 'Poland')
vic = self.tree.AppendItem(aus, 'Victoria')

```

Additional branches can be created by appending items to existing tree elements.

```

def OnCtrlT(self, e):

    rootItem = self.tree.GetRootItem()
    item, _ = self.tree.GetFirstChild(rootItem)
    self.Traverse(item)

```

The `OnCtrlT` method is called when we press the Ctrl+T keyboard shortcut. The root item is retrieved with the `GetRootItem` method. Now that we have the root item, we can find the first child of the root using the `GetFirstChild` method. The `Traverse` method is called to recursively find all the subitems.

```

def Traverse(self, item):

    while item.IsOk():

        print(self.tree.GetItemText(item))
        child, _ = self.tree.GetFirstChild(item)

        if child.IsOk():
            self.Traverse(child)

        item = self.tree.GetNextSibling(item)

```

The `Traverse` method uses a recursion to find all items of the tree control. First, all children of a particular top level item are found. The `IsOk` method checks if there are further child items. After that, we get a sibling with the `GetNextSibling` method. If there is some sibling found, we look for all its children and grandchildren etc.

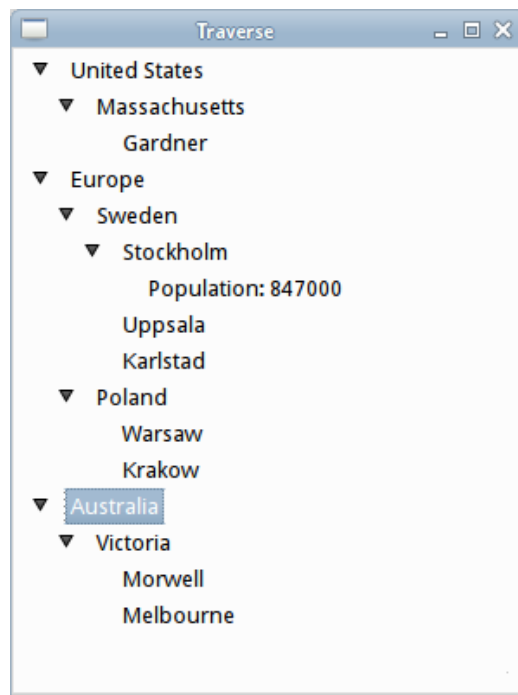


Figure 6.3: Traversing wx.TreeCtrl items

Figure 6.3 shows multiple branches of a `wx.TreeCtrl`, which are traversed in the code example.

6.4 Searching

In the next example we search for a text in the tree control. The case sensitivity of the text search can be controlled by a check box. Our task includes finding the text item, selecting it in the tree control, and making sure that it is visible.

Listing 6.4: Searching for text

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we search for a string
in a wx.TreeCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):
```

```

def __init__(self, *args, **kw):
    super(Example, self).__init__(*args, **kw)

    self.InitUI()
    self.InitData()

    self.SetSize((750, 450))
    self.SetTitle('Search')
    self.Centre()

def InitUI(self):

    vbox = wx.BoxSizer(wx.VERTICAL)

    self.tree = wx.TreeCtrl(self, style=wx.TR_HAS_BUTTONS
        |wx.TR_HIDE_ROOT|wx.TR_LINES_AT_ROOT)

    hbox = wx.BoxSizer(wx.HORIZONTAL)
    spnl = wx.Panel(self)
    spnl.SetSizer(hbox)

    csBox = wx.CheckBox(spnl, label="Case sensitive")
    csBox.SetValue(True)
    csBox.Bind(wx.EVT_CHECKBOX, self.SetCaseSensitive)
    hbox.Add(csBox, flag=wx.ALL, border=5)

    self.txt = wx.TextCtrl(spnl)

    hbox.Add(self.txt, flag=wx.ALL, border=5)

    sbtn = wx.Button(spnl, label="Search")
    sbtn.Bind(wx.EVT_BUTTON, self.OnButtonClick)
    hbox.Add(sbtn, flag=wx.ALL, border=5)

    vbox.Add(self.tree, proportion=1, flag=wx.EXPAND)
    vbox.Add(spnl)
    self.SetSizer(vbox)

def InitData(self):

    self.case = True

    root = self.tree.AddRoot("Root")

    cits = self.tree.AppendItem(root, 'Cities')
    adjs = self.tree.AppendItem(root, 'Adjectives')

    cities = ["Bratislava", "Budapest", "Warsaw",
        "Prague", "Kiev"]

    adjectives = ["Huge", "Great", "Formal", "Eloquent",
        "Daring", "Cute", "Big", "Alternative"]

    self.AddTreeNodes(adjs, adjectives)
    self.AddTreeNodes(cits, cities)

def AddTreeNodes(self, parentItem, items):

    for item in items:

```

```

        self.tree.AppendItem(parentItem, item)

def SetCaseSensitive(self, e):
    if e.IsChecked():
        self.case = True
    else:
        self.case = False

def OnButtonClick(self, e):
    value = self.txt.GetValue()
    rootItem = self.tree.GetRootItem()
    item, _ = self.tree.GetFirstChild(rootItem)
    self.Traverse(item, value)

def Traverse(self, item, value):
    while item.IsOk():
        titem = self.tree.GetItemText(item)

        if self.case:
            if titem == value:
                self.ItemFound(item)
                return
        else:
            if titem.lower() == value.lower():
                self.ItemFound(item)
                return

        child, _ = self.tree.GetFirstChild(item)

        if child.IsOk():
            self.Traverse(child, value)

        item = self.tree.GetNextSibling(item)

def ItemFound(self, item):
    self.tree.UnselectAll()
    self.tree.SelectItem(item, True)
    self.tree.EnsureVisible(item)

def main():
    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We have two branches in the tree control: cities and adjectives. At the bottom of the window there is a panel with a text control, a check box, and a button.

```
csBox = wx.CheckBox(spn1, label="Case sensitive")
```

This check box determines whether the search is case sensitive.

```
self.txt = wx.TextCtrl(spn1)
```

In this text control we write text that we search for.

```
cities = ["Bratislava", "Budapest", "Warsaw",
          "Prague", "Kiev"]
```

```
adjectives = ["Huge", "Great", "Formal", "Eloquent",
              "Daring", "Cute", "Big", "Alternative"]
```

The tree control holds these names of cities and adjectives.

```
def SetCaseSensitive(self, e):
```

```
    if e.IsChecked():
        self.case = True
    else:
        self.case = False
```

When we click on the check box, the `SetCaseSensitive` method is called. The state of the check box is stored in the `case` variable.

```
def OnButtonClick(self, e):
```

```
    value = self.txt.GetValue()
    rootItem = self.tree.GetRootItem()
    item, _ = self.tree.GetFirstChild(rootItem)
    self.Traverse(item, value)
```

The method is called when we click on the search button. We get the value from the text control. We find the root item and the first child of the root item. Finally, the `Traverse` method is called.

```
def Traverse(self, item, value):
    ...
```

In the `Traverse` method, we go through all the items of the tree control. A recursive algorithm finds all children of a particular item. After that we look for a sibling of that particular item and its children.

```
titem = self.tree.GetItemText(item)
```

```
if self.case:
    if titem == value:
        self.ItemFound(item)
        return
else:
    if titem.lower() == value.lower():
        self.ItemFound(item)
        return
```

We get the text of a current tree item. We compare it to the text from the text control; the case sensitivity is taken into account. If the two values match, we call the `ItemFound` method.

```
def ItemFound(self, item):

    self.tree.UnselectAll()
    self.tree.SelectItem(item, True)
    self.tree.EnsureVisible(item)
```

In the `ItemFound` method, we unselect all items, select the item that was found to match and ensure that it is visible on the tree control.

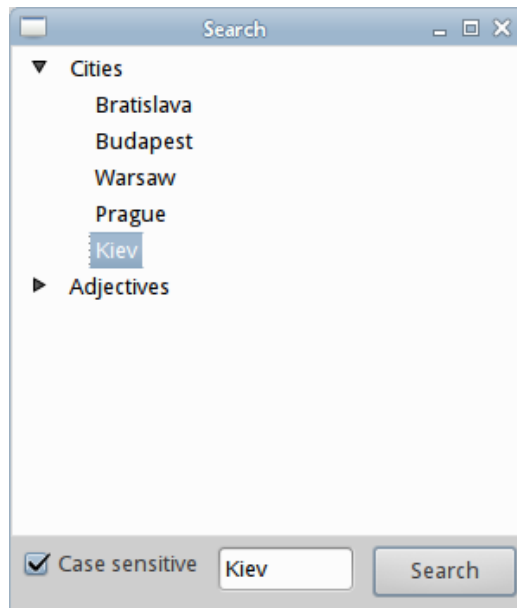


Figure 6.4: Searching for text

Figure 6.4 shows a `wx.TreeCtrl` whose item was selected after a successful search.

6.5 Sorting items

It is possible to sort items in a `wx.TreeCtrl`. To be able to do sorting, we must create a custom tree control. We have to implement the `OnCompareItems` method. In this method, we can control the sorting order (ascending/descending) and sorting type (numerical/string). The call of the `SortChildren` method starts the sorting of items in which the `OnCompareItems` method will be taken into account.

Listing 6.5: Sorting items

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this example we sort the items
in the wx.TreeCtrl.

Author: Jan Bodnar
```

Website: zetcode.com
"""

```
import wx
```

```
class MyTreeCtrl(wx.TreeCtrl):

    def __init__(self, *args, **kw):
        super(MyTreeCtrl, self).__init__(*args, **kw)

        self.order = 'asc'
        self.sortType = 'str'

    def OnCompareItems(self, item1, item2):

        def cmp(a, b):
            return (a > b) - (a < b)

        def cmp2(a, b):
            if int(a) > int(b):
                return 1
            elif int(a) < int(b):
                return -1
            else: return 0

        i1 = self.GetItemText(item1)
        i2 = self.GetItemText(item2)

        if self.order == 'asc' and self.sortType == 'str':
            print('h1')
            return cmp(i1, i2)

        elif self.order == 'desc' and self.sortType == 'str':
            print('h2')
            return cmp(i2, i1)

        elif self.order == 'asc' and self.sortType == 'num':
            return cmp2(i1, i2)

        elif self.order == 'desc' and self.sortType == 'num':
            return cmp2(i2, i1)

        else: return 0

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((700, 450))
        self.SetTitle('Sorting')
        self.Centre()

    def InitUI(self):
```



```

pnl = wx.Panel(self)
vbox = wx.BoxSizer(wx.VERTICAL)

self.tree = MyTreeCtrl(pnl, style=wx.TR_HAS_BUTTONS|
    wx.TR_HIDE_ROOT|wx.TR_LINES_AT_ROOT)

hbox = wx.BoxSizer(wx.HORIZONTAL)
buttons = wx.Panel(pnl)
buttons.SetSizer(hbox)

asc = wx.CheckBox(buttons, label="Ascending")
asc.SetValue(True)
asc.Bind(wx.EVT_CHECKBOX, self.SetOrder)

hbox.Add(asc, flag=wx.ALL, border=5)

num = wx.CheckBox(buttons, label="Numerical")
num.Bind(wx.EVT_CHECKBOX, self.SetSortType)
hbox.Add(num, flag=wx.ALL, border=5)

sortButton = wx.Button(buttons, label="Sort")
sortButton.Bind(wx.EVT_BUTTON, self.DoSort)
hbox.Add(sortButton, flag=wx.TOP|wx.BOTTOM, border=5)

vbox.Add(self.tree, proportion=1, flag=wx.EXPAND)
vbox.Add(buttons)
pnl.SetSizer(vbox)

def InitData(self):

    root = self.tree.AddRoot("Root")

    nums = self.tree.AppendItem(root, 'Numbers')
    strs = self.tree.AppendItem(root, 'Strings')

    numbers = ["12", "3", "54", "11", "8", "9", "0", "1"]

    strings = ["Huge", "Great", "Formal", "Eloquent",
        "Daring", "Cute", "Big", "Alternative"]

    self.AddTreeNodes(strs, strings)
    self.AddTreeNodes(nums, numbers)

def AddTreeNodes(self, parentItem, items):

    for item in items:
        self.tree.AppendItem(parentItem, item)

def SetOrder(self, e):

    if e.IsChecked():
        self.tree.order = 'asc'
    else:
        self.tree.order = 'desc'

def SetSortType(self, e):

```

```

        if e.IsChecked():
            self.tree.sortType = 'num'
        else:
            self.tree.sortType = 'str'

    def DoSort(self, e):

        item = self.tree.GetSelection()
        val = self.tree.GetItemText(item)

        if val == 'Strings' and self.tree.sortType == 'num':
            msg = 'Numerical sorting not supported'
            dial = wx.MessageDialog(None, msg,
                                    'Exclamation', wx.OK | wx.ICON_EXCLAMATION)
            dial.ShowModal()
        else:
            self.tree.SortChildren(item)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We have a tree control, two check boxes, and a button on the window. The two check boxes control the sorting order and type. The button starts the sorting. A top level item which should be sorted must be selected first.

```

class MyTreeCtrl(wx.TreeCtrl):

    def __init__(self, *args, **kw):
        super(MyTreeCtrl, self).__init__(*args, **kw)

        self.order = 'asc'
        self.sortType = 'str'

```

A custom tree control is created. The two variables hold the current sorting order and type.

```

def OnCompareItems(self, item1, item2):

    def cmp(a, b):
        return (a > b) - (a < b)

    def cmp2(a, b):
        if int(a) > int(b):
            return 1
        elif int(a) < int(b):
            return -1
        else: return 0

    i1 = self.GetItemText(item1)
    i2 = self.GetItemText(item2)

```

...

The `OnCompareItems` method has to be implemented in order to be able to sort tree items. We have two custom comparison methods: `cmp` for strings and `cmp2` for integers.

The method is used when the `SortChildren` method is called on the tree control. It takes two parameters, two tree items, which are compared and sorted accordingly. The method returns a negative value, zero, or a positive value. The `GetItemText` method retrieves the text of a tree item in question.

```
def DoSort(self, e):  
  
    item = self.tree.GetSelection()  
    val = self.tree.GetItemText(item)  
  
    if val == 'Strings' and self.tree.sortType == 'num':  
        msg = 'Numerical sorting not supported'  
        dial = wx.MessageDialog(None, msg,  
                                'Exclamation', wx.OK | wx.ICON_EXCLAMATION)  
        dial.ShowModal()  
    else:  
        self.tree.SortChildren(item)
```

The `DoSort` method is called when the sort button is clicked. We get the text of the parent item which we are about to sort. An error message is shown if we wanted to sort strings numerically. If all is OK, the `SortChildren` method is called. It sorts the tree items using the values from the `OnCompareItems` method.

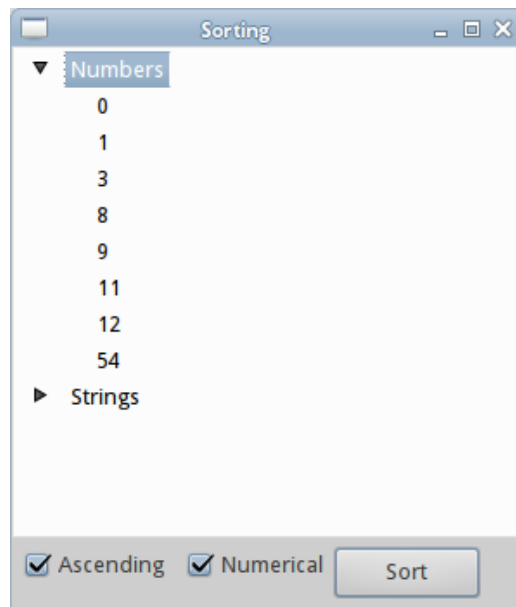


Figure 6.5: Sorting tree items

Figure 6.5 shows a `wx.TreeCtrl` which has numbers sorted numerically and in ascending order.

6.6 Images

It is possible to show images in a `wx.TreeCtrl`. Images are displayed before the text in the tree control.

Listing 6.6: Images

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we show images in a
wx.TreeCtrl.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()
        self.InitData()

        self.SetSize((750, 400))
        self.SetTitle('Images')
        self.Centre()

    def InitUI(self):

        self.tree = wx.TreeCtrl(self)

    def InitData(self):

        il = wx.ImageList(16, 16)

        imfld = wx.ArtProvider.GetBitmap(wx.ART_FOLDER,
                                         wx.ART_OTHER, (16, 16))
        fld_idx = il.Add(imfld)

        imfil = wx.ArtProvider.GetBitmap(wx.ART_NORMAL_FILE,
                                         wx.ART_OTHER, (16, 16))
        fil_idx = il.Add(imfil)

        self.tree.AssignImageList(il)

        root = self.tree.AddRoot('Sources')
        self.tree.SetItemImage(root, fld_idx)

        itm = self.tree.AppendItem(root, 'simple.py')
        self.tree.SetItemImage(itm, fil_idx)

        itm = self.tree.AppendItem(root, 'traverse.py')
        self.tree.SetItemImage(itm, fil_idx)
```

```

        itm = self.tree.AppendItem(root, 'selections.py')
        self.tree.SetItemImage(itm, fil_idx)

        itm = self.tree.AppendItem(root, 'search.py')
        self.tree.SetItemImage(itm, fil_idx)

        itm = self.tree.AppendItem(root, 'sort.py')
        self.tree.SetItemImage(itm, fil_idx)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We show two images in the example. One for the folder and the other for the files under the folder. Images are taken from the wxPython art provider.

```
il = wx.ImageList(16, 16)
```

An image list for images of 16x16 size is created.

```

imfld = wx.ArtProvider.GetBitmap(wx.ART_FOLDER,
    wx.ART_OTHER, (16, 16))
fld_idx = il.Add(imfld)

```

A folder image is retrieved from the art provider and added to the image list. The `Add` method also returns the image index.

```
self.tree.AssignImageList(il)
```

The image list is assigned to the tree using the `AssignImageList` method.

```
self.tree.SetItemImage(root, fld_idx)
```

The image is set to the tree item with `SetItemImage`. The method parameters are the tree item and the image index.

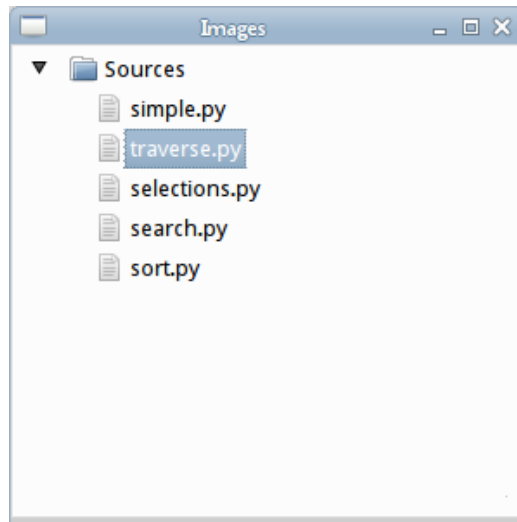


Figure 6.6: Images

Figure 6.6 shows images in a `wx.TreeCtrl`.

6.7 Lazy tree evaluation

Displaying a lot of data in a `wx.TreeCtrl` may consume large amounts of memory and can make the application sluggish. Therefore, a technique called lazy tree evaluation is used when working with many data items. It is similar to the virtual list control. In lazy tree evaluation, we only add tree items at the moment when they are needed, thus reducing the memory usage and application response time.

Listing 6.7: Lazy tree evaluation

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program we create lazy tree
evaluation example.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx, os

class LazyTree(wx.TreeCtrl):

    def __init__(self, *args, **kw):
        super(LazyTree, self).__init__(*args, **kw)

        self.InitUI()
```

```

def InitUI(self):

    self.Bind(wx.EVT_TREE_ITEM_EXPANDING,
              self.OnExpandItem)

    self.homedir = os.path.expanduser('~')

    root = self.AddRoot(self.homedir)
    self.SetItemData(root, self.homedir)
    self.SetItemHasChildren(root)

def OnExpandItem(self, e):

    item = e.GetItem()
    cpath = self.GetItemData(item)

    sdr = os.listdir(cpath)
    sdr.sort()

    for child in sdr:

        path = os.path.join(cpath, child)

        if not child.startswith('.'):

            chid = self.AppendItem(item, child)
            self.SetItemData(chid, path)

            if os.path.isdir(path):
                l = os.listdir(path)
                if l:
                    self.SetItemHasChildren(chid)

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

        self.SetSize((750, 450))
        self.SetTitle('Lazy evaluation')
        self.Centre()

    def InitUI(self):

        LazyTree(self)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":

```

```
main()
```

We browse the home directory of a user. We add items to the tree control at the moment of expanding a tree node.

```
self.Bind(wx.EVT_TREE_ITEM_EXPANDING,
          self.OnExpandItem)
```

The event name for three expansion is `wx.EVT_TREE_ITEM_EXPANDING`.

```
self.homedir = os.path.expanduser('~')
```

In the example, we display directories and files of a home directory of a user.

```
root = self.AddRoot(self.homedir)
self.SetItemData(root, self.homedir)
self.SetItemHasChildren(root)
```

We add the root item to the tree control with the `AddRoot` method. The `SetItemData` associates an object with a tree item. In our case, we add a path to each tree item. This path is used later to add tree items when we expand a tree node. The `SetItemHasChildren` forces the appearance of the expand button. In this case we assume that there is at least one file or directory in the home directory and therefore it can be expanded.

```
def OnExpandItem(self, e):
    item = e.GetItem()
    cpath = self.GetItemData(item)

    sdr = os.listdir(cpath)
    sdr.sort()
    ...
```

The `OnExpandItem` method is called when we have clicked on a tree item. We get the expanding tree item and the associated path. The `listdir` Python function returns all the files and directories under the given path. The entries are sorted using the `sort` method.

```
for child in sdr:

    path = os.path.join(cpath, child)

    if not child.startswith('.'):

        chid = self.AppendItem(item, child)
        self.SetItemData(chid, path)

        if os.path.isdir(path):
            l = os.listdir(path)
            if l:
                self.SetItemHasChildren(chid)
```

We append all the entries to the expanding tree node. We also associate a path to each new tree item. If the item is a directory and is not empty, we display an expand button to the left of the tree item. We do not display files that start

with a dot character; these are hidden files on Unix systems.

Chapter 7

wx.grid.Grid

A `wx.grid.Grid` is a complex widget for working with tabular data; it is also called a table widget. It has columns and rows. The intersections of columns and rows are called cells. In these cells we write data. A grid widget was popularized by spreadsheet applications such as Excel or Open Office Calc.

There are several supporting objects for the grid widget. Cell attributes control the appearance of the cells in the grid. Cell editors determine the way of how the data is going to be modified in a cell. A cell editor can be a text control, a spin box, or a choice box. Cell renderers are objects responsible for drawing cells.

With the help of the `wx.grid.GridTableBase`, we can construct our grid with a Model & view desing pattern.

7.1 Basics

The first example shows some basics of a `wx.grid.Grid` widget.

Listing 7.1: Basics

```
def CreateGrid(self):

    grid = wx.grid.Grid(self)
    grid.CreateGrid(10, 10)

    grid.SetCellValue(0, 0, "Python")
    grid.SetCellValue(0, 3, "Java")
    grid.SetCellValue(3, 3, "Ruby")
    grid.SetCellValue(3, 0, "PHP")

    grid.SetCellTextColour(0, 0, wx.RED)
    grid.SetCellBackgroundColour(3, 3, "GREY")
    font = wx.Font(11, wx.FONTFAMILY_SWISS,
                   wx.NORMAL, wx.FONTWEIGHT_BOLD)
    grid.SetCellFont(0, 3, font)

    grid.SetCellSize(1, 1, 1, 3)
    grid.SetCellValue(1, 1,
                     "ZetCode, tutorials for programmers")
```

We create a grid widget with ten columns and rows. We insert some text into a few cells and modify some attributes of cells.

```
grid = wx.grid.Grid(self)
grid.CreateGrid(10, 10)
```

A grid widget is created. The `CreateGrid` method constructs ten rows and ten columns.

```
grid.SetCellValue(0, 0, "Python")
```

With the `SetCellValue` method, we insert a text at a specified cell. The first two parameters are the row and column numbers. In our case, we write "Python" text into the top-left corner cell of the grid widget.

```
grid.SetCellTextColour(0, 0, wx.RED)
```

We modify the text colour in the top-left cell with `SetCellTextColour`.

```
grid.SetCellBackgroundColour(3, 3, "GREY")
```

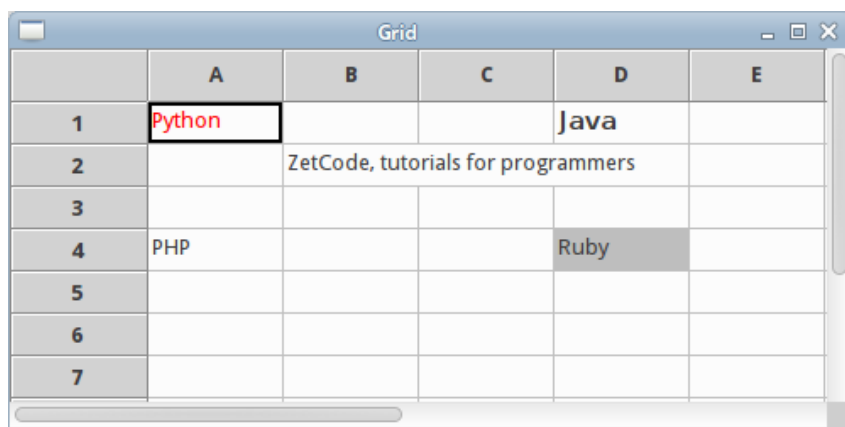
The `SetCellBackgroundColour` method is used to set the background colour of a cell to gray.

```
font = wx.Font(11, wx.FONTFAMILY_SWISS,
               wx.NORMAL, wx.FONTWEIGHT_BOLD)
grid.SetCellFont(0, 3, font)
```

We change a font for the specified cell with `SetCellFont`.

```
grid.SetCellSize(1, 1, 1, 3)
grid.SetCellValue(1, 1,
                 "ZetCode, tutorials for programmers")
```

The `SetCellSize` method changes the size of a cell. Our cell will expand three cells horizontally. The cell was expanded in order for the text to fit the cell.



	A	B	C	D	E
1	Python			Java	
2		ZetCode, tutorials for programmers			
3					
4	PHP			Ruby	
5					
6					
7					

Figure 7.1: wx.grid.Grid basics

Figure 7.1 shows some basic functionality of a `wx.grid.Grid` widget.

7.2 Cell attributes

The `wx.grid.GridCellAttr` class encapsulates the characteristics of a grid cell such as text, background colour, font, or alignment. With the help of the grid cell attributes, we can change the appearance and the functionality of a grid cell.

Listing 7.2: Grid cell attributes

```
def CreateGrid(self):

    grid = wx.grid.Grid(self)
    grid.CreateGrid(10, 10)

    grid.SetCellValue(0, 0, "Jane")
    grid.SetCellValue(0, 1, "David")

    attr1 = wx.grid.GridCellAttr()
    attr1.SetAlignment(wx.ALIGN_CENTRE, wx.ALIGN_CENTRE)
    attr1.SetTextColour("blue")
    attr1.SetBackgroundColour("gray")
    attr1.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL,
                          wx.BOLD))
    grid.SetAttr(0, 0, attr1)

    attr2 = wx.grid.GridCellAttr()
    attr2.SetReadOnly()
    grid.SetAttr(0, 1, attr2)
```

In the above code example, we use two different cell attributes.

```
attr1 = wx.grid.GridCellAttr()
```

An instance of a grid cell attribute is created..

```
attr1.SetAlignment(wx.ALIGN_CENTRE, wx.ALIGN_CENTRE)
attr1.SetTextColour("blue")
attr1.SetBackgroundColour("gray")
attr1.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL,
                      wx.BOLD))
```

We set the alignment, text and background colour, and font of the grid cell attribute.

```
grid.SetAttr(0, 0, attr1)
```

The grid cell attribute is applied for the top-left cell of the grid.

```
attr2 = wx.grid.GridCellAttr()
attr2.SetReadOnly()
grid.SetAttr(0, 1, attr2)
```

In the second case, we make the cell readonly with `SetReadOnly`.

7.3 Show hide example

It is possible to toggle the visibility of the grid lines and the column and row labels. The following example will demonstrate this.

Listing 7.3: Show hide example

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this example, we toggle the visibility
of row, column labels and grid lines.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.grid

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateMenuBar()
        self.CreateGrid()

        self.SetSize((500, 350))
        self.SetTitle('Grid')
        self.Centre()

    def CreateMenuBar(self):

        ID_SHOW_COLS = wx.NewIdRef()
        ID_SHOW_ROWS = wx.NewIdRef()
        ID_SHOW_LINES = wx.NewIdRef()

        mb = wx.MenuBar()

        fileMenu = wx.Menu()
        fileMenu.AppendCheckItem(ID_SHOW_COLS,
                                'Show Column Labels')
        fileMenu.Check(ID_SHOW_COLS, True)
        fileMenu.AppendCheckItem(ID_SHOW_ROWS,
                                'Show Row Labels')
        fileMenu.Check(ID_SHOW_ROWS, True)
        fileMenu.AppendCheckItem(ID_SHOW_LINES,
                                'Show lines')
        fileMenu.Check(ID_SHOW_LINES, True)

        mb.Append(fileMenu, '&File')
```

```

        self.SetMenuBar(mb)

        self.Bind(wx.EVT_MENU, self.ToggleColLabels,
                   id=ID_SHOW_COLS)
        self.Bind(wx.EVT_MENU, self.ToggleRowLabels,
                   id=ID_SHOW_ROWS)
        self.Bind(wx.EVT_MENU, self.ToggleLines,
                   id=ID_SHOW_LINES)

    def CreateGrid(self):

        self.grid = wx.grid.Grid(self)
        self.grid.CreateGrid(7, 5)

        self.w = self.grid.GetRowLabelSize()
        self.h = self.grid.GetColLabelSize()

    def ToggleColLabels(self, e):

        sel = e.GetSelection()

        if sel:
            self.grid.SetColLabelSize(self.h)
        else:
            self.grid.HideColLabels()

    def ToggleRowLabels(self, e):

        sel = e.GetSelection()

        if sel:
            self.grid.SetRowLabelSize(self.w)
        else:
            self.grid.HideRowLabels()

    def ToggleLines(self, e):

        self.grid.EnableGridLines(e.GetSelection())

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We have a menubar with menu items to show/hide the grid lines and column and row labels.

```

self.CreateMenuBar()
self.CreateGrid()

```

The creation of the menubar is delegated to the `CreateMenuBar` method. The creation of the grid is delegated to the `CreateGrid` method.

```
def CreateMenuBar(self):  
  
    ID_SHOW_COLS = wx.NewIdRef()  
    ID_SHOW_ROWS = wx.NewIdRef()  
    ID_SHOW_LINES = wx.NewIdRef()  
  
    mb = wx.MenuBar()  
    ...
```

We generate three ids for three menu items.

```
fileMenu = wx.Menu()  
fileMenu.AppendCheckItem(ID_SHOW_COLS,  
    'Show Column Labels')  
fileMenu.Check(ID_SHOW_COLS, True)
```

The menu items are check items; they are created with `AppendCheckItem`. The check mark indicates whether the object is visible.

```
def CreateGrid(self):  
  
    self.grid = wx.grid.Grid(self)  
    self.grid.CreateGrid(7, 5)  
  
    self.w = self.grid.GetRowLabelSize()  
    self.h = self.grid.GetColLabelSize()
```

In the `CreateGrid` method, we create a grid with seven rows and five columns. We get the size of the row and column labels. These will be needed later.

```
def ToggleColLabels(self, e):  
  
    sel = e.GetSelection()  
  
    if sel:  
        self.grid.SetColLabelSize(self.h)  
    else:  
        self.grid.HideColLabels()
```

In the `ToggleColLabels` method, we toggle the visibility of the column labels. We get the state of the corresponding check menu item with the `GetSelection` method.

Depending on its state, we show the column labels with the `SetColLabelSize` method using the saved height value as a parameter or hide the column labels with the `HideColLabels` method.

```
def ToggleRowLabels(self, e):  
  
    sel = e.GetSelection()  
  
    if sel:  
        self.grid.SetRowLabelSize(self.w)  
    else:  
        self.grid.HideRowLabels()
```

Showing, hiding row labels is similar. We use the `SetRowLabelSize` method to show the row labels and the `HideRowLabels` method to hide them.

```
def ToggleLines(self, e):  
    self.grid.EnableGridLines(e.GetSelection())
```

Finally, the grid lines are enabled or disabled with the `EnableGridLines` method.

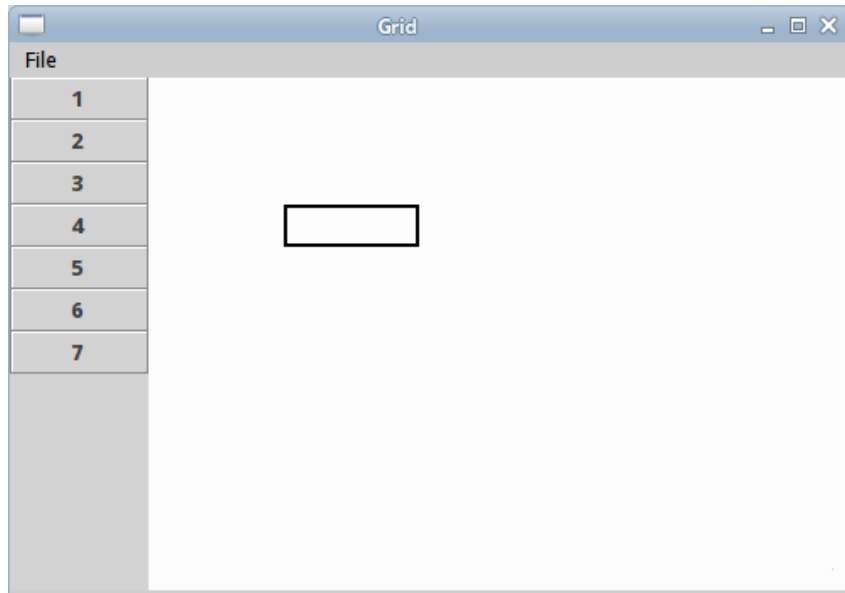


Figure 7.2: Show hide example

Figure 7.2 shows a `wx.grid.Grid` with hidden column labels and grid lines.

7.4 Moving the cursor

In most spreadsheet applications when we click on a column/row header, the whole column/row is highlighted and the cursor is moved to the highlighted column/row. By default, the grid cursor does not move in `wx.grid.Grid` in such situations. In the following example, we show how we can achieve this.

Listing 7.4: Moving the cursor

```
def InitUI(self):  
    self.CreateGrid()  
  
    self.SetSize((700, 450))  
    self.SetTitle('Grid')  
    self.Centre()  
  
def CreateGrid(self):
```



```

self.grid = wx.grid.Grid(self)
self.grid.CreateGrid(10, 10)

self.grid.Bind(wx.grid.EVT_GRID_LABEL_LEFT_CLICK,
               self.OnLabelLeftClick)

def OnLabelLeftClick(self, e):

    col = e.GetCol()
    row = e.GetRow()

    if col >= 0:
        row = 0
    else:
        col = 0

    self.grid.SetGridCursor(row, col)

```

In the example, the cursor is moved to the first cell of the highlighted row or column.

```

self.grid.Bind(wx.grid.EVT_GRID_LABEL_LEFT_CLICK,
               self.OnLabelLeftClick)

```

The `wx.grid.EVT_GRID_LABEL_LEFT_CLICK` event is used to react to the left mouse click on a grid label. This applies to both row and column labels.

```

def OnLabelLeftClick(self, e):

    col = e.GetCol()
    row = e.GetRow()
    ...

```

The second parameter of the `OnLabelLeftClick` method is a `wx.GridEvent` which holds the information of a triggered grid event. The `GetCol` method of the event object returns the column number at which the event happened. Likewise, we get the row number with the `GetRow` method. Since the event does not distinguish between row and column labels, one of the returned values is always -1.

```

if col >= 0:
    row = 0
else:
    col = 0

self.grid.SetGridCursor(row, col)

```

If we have clicked on the column label, the row number is set to 0. And vice versa. The cursor is moved to the cell with the `SetGridCursor` method.

```

e.Skip()

```

The `Skip` method sends the event for further built-in processing. Without this method call, the cells would not be highlighted.

7.5 Enter key

By default, the grid cursor moves down when we hit the Enter key. The direction of the cursor can be changed. When we write a lot of data in rows, we might want the grid cursor move to the right.

Listing 7.5: Enter key

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this example, we change the direction
of the grid cursor for the Enter key.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.grid

class m:

    ID_MOVE_RIGHT = wx.NewIdRef()
    ID_MOVE_DOWN = wx.NewIdRef()

    ENTER_KEY_MOVE_RIGHT = 'right'
    ENTER_KEY_MOVE_DOWN = 'down'

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateMenuBar()
        self.CreateGrid()

        self.SetSize((700, 450))
        self.SetTitle('Grid')
        self.Centre()

    def CreateMenuBar(self):

        mb = wx.MenuBar()

        inMenu = wx.Menu()
        inMenu.AppendRadioItem(m.ID_MOVE_DOWN, 'Move down')
        inMenu.AppendRadioItem(m.ID_MOVE_RIGHT, 'Move right')

        mb.Append(inMenu, '&Input')
        self.SetMenuBar(mb)
```

```

        self.Bind(wx.EVT_MENU, self.SetInput,
                    id=m.ID_MOVE_RIGHT)
        self.Bind(wx.EVT_MENU, self.SetInput,
                    id=m.ID_MOVE_DOWN)

    def SetInput(self, e):

        if e.GetId() == m.ID_MOVE_RIGHT:
            self.moveDirection = m.ENTER_KEY_MOVE_RIGHT
        else:
            self.moveDirection = m.ENTER_KEY_MOVE_DOWN

    def OnKeyDown(self, e):

        key = e.GetKeyCode()

        if key == wx.WXK_RETURN:

            if self.moveDirection == m.ENTER_KEY_MOVE_RIGHT:
                self.grid.MoveCursorRight(e.ShiftDown())
            else:
                self.grid.MoveCursorDown(e.ShiftDown())
        else:
            e.Skip()

    def CreateGrid(self):

        self.moveDirection = m.ENTER_KEY_MOVE_DOWN
        self.grid = wx.grid.Grid(self)
        self.grid.CreateGrid(7, 6)

        self.grid.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

There is an Input menu where we have two menu items. They control the movement of the grid cursor after pressing the enter key.

```

class m:

    ID_MOVE_RIGHT = wx.NewIdRef()
    ID_MOVE_DOWN = wx.NewIdRef()

    ENTER_KEY_MOVE_RIGHT = 'right'
    ENTER_KEY_MOVE_DOWN = 'down'

```

We have our constants defined in the `m` class to avoid global values.

```
inMenu = wx.Menu()
inMenu.AppendRadioItem(m.ID_MOVE_DOWN, 'Move down')
inMenu.AppendRadioItem(m.ID_MOVE_RIGHT, 'Move right')
```

The menu items are radio items created with `AppendRadioItem`;

```
def SetInput(self, e):
    if e.GetId() == m.ID_MOVE_RIGHT:
        self.MoveDirection = m.ENTER_KEY_MOVE_RIGHT
    else:
        self.MoveDirection = m.ENTER_KEY_MOVE_DOWN
```

In the `SetInput` method, we set the direction movement of the grid cursor.

```
def OnKeyDown(self, e):
    key = e.GetKeyCode()

    if key == wx.WXK_RETURN:
        if self.moveDirection == m.ENTER_KEY_MOVE_RIGHT:
            self.grid.MoveCursorRight(e.ShiftDown())
        else:
            self.grid.MoveCursorDown(e.ShiftDown())
    else:
        e.Skip()
```

In the `OnKeyDown` method, we first get the key code. The key code for the enter key is `wx.WXK_RETURN`. (The Enter key is also called the Return key.) We check the `moveDirection` variable and depending on its value, we call either the `MoveCursorRight` or `MoveCursorDown` method. They take an `expand` selection parameter which determines whether the potential selection will be expanded. In our case, it will be expanded if a Shift key was held simultaneously.

7.6 Selections

Cells in a `wx.grid.Grid` widget can be selected. Selected cells have different background colors. There are three types of selections: individual selections, block selections, and full column/full row selections. Individual selections are performed with a mouse pointer and a cursor key; arbitrary cells can be selected in this way.

Individual selections are found by the `GetSelectedCells` method. Block selections are continuous selected cells chosen by a mouse pointer or by shift and arrow keys. These cells are found by the `GetSelectionBlockTopLeft` and `GetSelectionBlockBottomRight` methods. Full column and full row selections are performed by clicking on the headers of the columns and rows. Selected columns are found by the `GetSelectedCols` method and selected rows by the `GetSelectedRows` method. Selections are divided into these groups for efficiency reasons.

Listing 7.6: Selections

```
#!/usr/bin/env python
```

```

"""
ZetCode Advanced wxPython tutorial

In this program we work with selections
in a wx.grid.Grid.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.grid

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateGrid()

        self.SetSize((700, 450))
        self.SetTitle('Grid')
        self.Centre()

    def CreateGrid(self):

        self.grid = wx.grid.Grid(self)
        self.grid.CreateGrid(10, 10)

        self.grid.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)

        self.grid.SetFocus()

    def OnKeyDown(self, e):

        key = e.GetKeyCode()

        if key == wx.WXK_DELETE:

            cells = self.FindCellsToDelete()

            for row, col in cells:
                self.grid.SetCellValue(row, col, '')

            return

        e.Skip()

    def FindCellsToDelete(self):

        cells = []

        col = self.grid.GetGridCursorCol()

```

```

        row = self.grid.GetGridCursorRow()

        cells.append((row, col))

    sng = self.grid.GetSelectedCells()

    if len(sng) > 0:
        for row, col in sng:
            cells.append((row, col))

    whr = self.grid.GetSelectedRows()

    if len(whr) > 0:

        ncols = self.grid.GetNumberCols()

        for row in whr:
            for col in range(ncols):
                cells.append((row, col))

    whc = self.grid.GetSelectedCols()

    if len(whc) > 0:
        nrows = self.grid.GetNumberRows()

        for col in whc:
            for row in range(nrows):
                cells.append((row, col))

    tl = self.grid.GetSelectionBlockTopLeft()
    br = self.grid.GetSelectionBlockBottomRight()

    if len(tl) > 0:
        for col in range(tl[0][1], br[0][1] + 1):
            for row in range(tl[0][0], br[0][0] + 1):
                cells.append((row, col))

    return cells

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

The delete keyboard button erases all data from the selected cells in a grid. We take into account all types of selections.

```

if key == wx.WXK_DELETE:

    cells = self.FindCellsToDelete()

```

```

        for row, col in cells:
            self.grid.SetCellValue(row, col, '')

    return

```

If we press the delete keyboard button, we find all selected cells using the `FindCellsToDelete` method. We go through all these cells and delete their contents. The deletion is performed by setting empty strings to the respective cells.

```

def FindCellsToDelete(self):

    cells = []
    ...

```

The `FindCellsToDelete` method finds all cells whose data is going to be deleted. These are going to be selected cells and the cell, which has the active grid cursor. We put these cells into the `cells` list.

```

col = self.grid.GetGridCursorCol()
row = self.grid.GetGridCursorRow()

cells.append((row, col))

```

We get the column and row number of the active cell with `GetGridCursorCol` and `GetGridCursorRow`.

```

sng = self.grid.GetSelectedCells()

if len(sng) > 0:
    for row, col in sng:
        cells.append((row, col))

```

Here we add all the individually selected cells to the `cells` list. The selected cells are retrieved with `GetSelectedCells`.

```

whr = self.grid.GetSelectedRows()

if len(whr) > 0:

    ncols = self.grid.GetNumberCols()

    for row in whr:
        for col in range(ncols):
            cells.append((row, col))

```

We append all cells belonging to the fully selected rows. We get the selected rows with `GetSelectedRows`. The number of columns is determined with `GetNumberCols`.

```

whc = self.grid.GetSelectedCols()

if len(whc) > 0:
    nrows = self.grid.GetNumberRows()

    for col in whc:
        for row in range(nrows):
            cells.append((row, col))

```

Here we append all cells belonging to fully selected columns. We get the selected columns with `GetSelectedCols`. The number of rows is determined with `GetNumberRows`.

```
tl = self.grid.GetSelectionBlockTopLeft()
br = self.grid.GetSelectionBlockBottomRight()

if len(tl) > 0:
    for col in range(tl[0][1], br[0][1] + 1):
        for row in range(tl[0][0], br[0][0] + 1):
            cells.append((row, col))
```

These lines append all block selections to the `cells` list.

7.7 Merging cells

Cells in a `wx.grid.Grid` widget can be merged. Merging of cells is performed with the `SetCellSize` method. A cell can span multiple rows or columns. The cells are unmerged by setting the size of the enlarged cell to 1 row/column.

Listing 7.7: Merging cells

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this example, we will merge and unmerge
cells in wx.grid.Grid widget.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
from wx.core import ID_STRIKETHROUGH
import wx.grid

class m:

    ID_SINGLE_CELL = wx.NewIdRef()
    ID_SPAN_CELL = wx.NewIdRef()
    ID_MERGE_CELLS = wx.NewIdRef()
    ID_UNMERGE_CELLS = wx.NewIdRef()

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateMenuBar()
        self.CreateGrid()
```



```

self.SetSize((700, 450))
self.SetTitle('Merging cells')
self.Centre()

def CreateMenuBar(self):

    mb = wx.MenuBar()

    self.eMenu = wx.Menu()
    self.eMenu.Append(m.ID_MERGE_CELLS,
                      'Merge Cells')
    self.eMenu.Append(m.ID_UNMERGE_CELLS,
                      'Unmerge Cells')

    mb.Append(self.eMenu, '&Edit')

    self.SetMenuBar(mb)

    self.Bind(wx.EVT_MENU, self.MergeCells,
              id=m.ID_MERGE_CELLS)
    self.Bind(wx.EVT_MENU, self.UnMergeCells,
              id=m.ID_UNMERGE_CELLS)
    self.Bind(wx.EVT_MENU_OPEN,
              self.OnMenuSelected)

def OnMenuSelected(self, e):

    tl = self.grid.GetSelectionBlockTopLeft()

    row = self.grid.GetGridCursorRow()
    col = self.grid.GetGridCursorCol()

    size = self.grid.GetCellSize(row, col)

    if size[0] == m.ID_SPAN_CELL:
        self.eMenu.Enable(m.ID_UNMERGE_CELLS, True)
    else:
        self.eMenu.Enable(m.ID_UNMERGE_CELLS, False)

    if len(tl) > 0:
        self.eMenu.Enable(m.ID_MERGE_CELLS, True)
    else:
        self.eMenu.Enable(m.ID_MERGE_CELLS, False)

def MergeCells(self, e):

    tl = self.grid.GetSelectionBlockTopLeft()
    br = self.grid.GetSelectionBlockBottomRight()

    if len(tl) > 0:

        w = br[0][1] - tl[0][1] + 1
        h = br[0][0] - tl[0][0] + 1

        self.grid.SetCellSize(tl[0][0], tl[0][1], h, w)
        self.grid.Refresh()

```

```

def UnMergeCells(self, e):

    row = self.grid.GetGridCursorRow()
    col = self.grid.GetGridCursorCol()

    size = self.grid.GetCellSize(row, col)

    if size[0] != m.ID_SINGLE_CELL:
        self.grid.SetCellSize(row, col, 1, 1)
        self.grid.Refresh()

def CreateGrid(self):

    self.grid = wx.grid.Grid(self)
    self.grid.CreateGrid(10, 10)

    self.grid.SetCellValue(1, 1,
        "ZetCode, tutorials for programmers")

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We have a menubar with two menu items. One merges cells, the other will unmerge them. These menu items are dynamically enabled or disabled, depending on whether there are cells to merge or unmerge.

```

class m:

    ID_SINGLE_CELL = wx.NewIdRef()
    ID_SPAN_CELL = wx.NewIdRef()
    ID_MERGE_CELLS = wx.NewIdRef()
    ID_UNMERGE_CELLS = wx.NewIdRef()

```

In order to avoid global variables, the menu ids are put into a separate class.

```

mb = wx.MenuBar()

self.eMenu = wx.Menu()
self.eMenu.Append(m.ID_MERGE_CELLS,
    'Merge Cells')
self.eMenu.Append(m.ID_UNMERGE_CELLS,
    'Unmerge Cells')

mb.Append(self.eMenu, '&Edit')

self.SetMenuBar(mb)

```

We create a menubar with two menu items.

```

self.Bind(wx.EVT_MENU, self.MergeCells,

```

```

        id=m.ID_MERGE_CELLS)
self.Bind(wx.EVT_MENU, self.UnMergeCells,
        id=m.ID_UNMERGE_CELLS)
self.Bind(wx.EVT_MENU_OPEN,
        self.OnMenuSelected)

```

Here we bind three events to three methods. The `wx.EVT_MENU_OPEN` event is generated when the menu is about to be opened.

```

def OnMenuSelected(self, e):
    ...

```

The `OnMenuSelected` method is called before the menu is displayed. In this method, we enable or disable the menu items. It depends on whether we can merge or unmerge some cells.

```

size = self.grid.GetCellSize(row, col)

if size[0] != 1 or size[1] != 1:
    self.eMenu.Enable(m.ID_UNMERGE_CELLS, True)
else:
    self.eMenu.Enable(m.ID_UNMERGE_CELLS, False)

```

We get the cell size with the `GetCellSize` method. If the cell is not a single cell, we enable the unmerge menu item. On the other hand if the cell is a single cell, the unmerge menu item is disabled.

```

def MergeCells(self, e):

    tl = self.grid.GetSelectionBlockTopLeft()
    br = self.grid.GetSelectionBlockBottomRight()

    if len(tl) > 0:

        w = br[0][1] - tl[0][1] + 1
        h = br[0][0] - tl[0][0] + 1

        self.grid.SetCellSize(tl[0][0], tl[0][1], h, w)
        self.grid.Refresh()

```

The cells are merged in the `MergeCells` method. We get the top-left and bottom-right cells of the block selection. From these values, we compute the width and height of the selection. The `SetCellSize` method merges the cells. Note that the contents of the top-left cell are shown in the newly formed cell. The contents of other cells are hidden.

```

def UnMergeCells(self, e):

    row = self.grid.GetGridCursorRow()
    col = self.grid.GetGridCursorCol()

    size = self.grid.GetCellSize(row, col)

    if size[0] != 1 or size[1] != 1:
        self.grid.SetCellSize(row, col, 1, 1)
        self.grid.Refresh()

```

In the `UnMergeCells` method, the cells are unmerged. If the size of a cell is two or

more rows/columns than we have a merged cell. The merging of cells is undone by setting the size of the enlarged cell to 1 row/column.

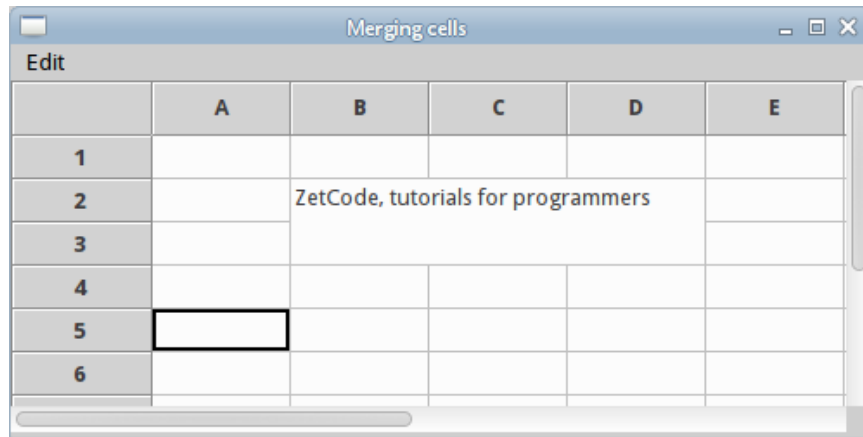


Figure 7.3: Merging cells

Figure 7.3 shows a `wx.grid.Grid` having several cells merged. The text of the top-left cell of the covered area is displayed.

7.8 Cell editors

Each cell in a `wx.grid.Grid` widget has a cell editor associated with it. The cell editor is responsible for data input. There are several predefined cell editors available. The `SetCellEditor` method is used to set a new cell editor for a specified cell.

Listing 7.8: Cell editors

```
def InitUI(self):
    self.CreateGrid()

    self.SetSize((700, 450))
    self.SetTitle('Cell editors')
    self.Centre()

def CreateGrid(self):
    grid = wx.grid.Grid(self)
    grid.CreateGrid(1, 4)

    grid.SetCellEditor(0, 0, wx.grid.GridCellTextEditor())
    grid.SetCellEditor(0, 1,
        wx.grid.GridCellNumberEditor(1, 100))
    grid.SetCellEditor(0, 2, wx.grid.GridCellBoolEditor())

    langs = ['Python', 'Ruby', 'Perl', 'PHP']
    grid.SetCellEditor(0, 3,
```

```
wx.grid.GridCellChoiceEditor(langs, False))
```

We have a grid with four cells. Each cell has a different cell editor.

```
grid.SetCellEditor(0, 0, wx.grid.GridCellTextEditor())
```

The first cell has the cell text editor. It is used for editing textual data.

```
grid.SetCellEditor(0, 1,  
    wx.grid.GridCellNumberEditor(1, 100))
```

The second cell has the cell number editor. The parameters are the min and max values. This editor shows a spin box which is used to select a number for the cell.

```
grid.SetCellEditor(0, 2, wx.grid.GridCellBoolEditor())
```

The third cell holds the cell boolean editor. A check box is shown while editing the cell. The user can select between two values: True or False.

```
langs = ['Python', 'Ruby', 'Perl', 'PHP']
```

```
grid.SetCellEditor(0, 3,  
    wx.grid.GridCellChoiceEditor(langs, False))
```

The last cell has the cell choice editor. This editor displays a choice box with options to select.

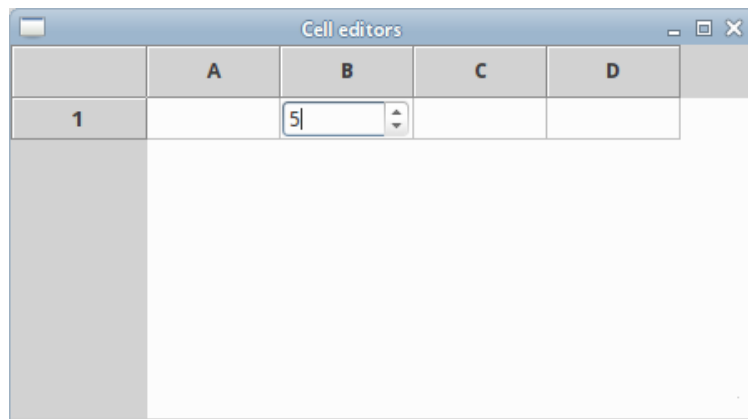


Figure 7.4: Cell editors

Figure 7.4 shows the `wx.grid.GridCellNumberEditor` in action.

7.9 Cell renderers

A cell renderer is an object which renders the cell contents. There are several built-in grid cell renderers:

- `wx.grid.GridCellStringRenderer`—formats strings

- `wx.grid.GridCellNumberRenderer`—formats integers
- `wx.grid.GridCellFloatRenderer`—formats floating point numbers
- `wx.grid.GridCellBoolRenderer`—formats boolean values
- `wx.grid.GridCellDateRenderer`—formats dates and times

Listing 7.9: Cell renderers

```
def CreateGrid(self):

    grid = wx.grid.Grid(self)
    grid.CreateGrid(1, 5)

    r1 = wx.grid.GridCellStringRenderer()
    r2 = wx.grid.GridCellNumberRenderer()
    r3 = wx.grid.GridCellFloatRenderer()
    r3.SetPrecision(2)
    r4 = wx.grid.GridCellBoolRenderer()
    r5 = wx.grid.GridCellDateRenderer()

    grid.SetCellRenderer(0, 0, r1)
    grid.SetCellRenderer(0, 1, r2)
    grid.SetCellRenderer(0, 2, r3)
    grid.SetCellRenderer(0, 3, r4)
    grid.SetCellRenderer(0, 4, r5)
```

We have five cells in a grid; each of them has a different cell renderer.

```
r1 = wx.grid.GridCellStringRenderer()
r2 = wx.grid.GridCellNumberRenderer()
r3 = wx.grid.GridCellFloatRenderer()
r3.SetPrecision(2)
r4 = wx.grid.GridCellBoolRenderer()
r5 = wx.grid.GridCellDateRenderer()
```

We create four cell renderers. They are used to render text, numbers, floating point values and boolean values.

```
grid.SetCellRenderer(0, 0, r1)
grid.SetCellRenderer(0, 1, r2)
grid.SetCellRenderer(0, 2, r3)
grid.SetCellRenderer(0, 3, r4)
```

The `SetCellRenderer` method sets a cell renderer for the specified cell.

7.10 Custom cell renderer

Built-in cell renderers meet most programming requirements. Sometimes we might want a special cell renderer. It is possible to create a custom cell renderer. Custom cell renderers are based on the `wx.grid.GridCellRenderer` class.

Listing 7.10: Custom cell renderer

```

#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this program, we build a custom cell renderer.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.grid

class MyImageRenderer(wx.grid.GridCellRenderer):

    def __init__(self, img, *args, **kw):
        super(MyImageRenderer, self).__init__(*args, **kw)

        self.img = img

    def Draw(self, grid, attr, dc, rect,
             row, col, isSelected):

        img = wx.MemoryDC()
        img.SelectObject(self.img)

        w, h = self.img.GetWidth(), self.img.GetHeight()

        dc.Blit(rect.x + 1, rect.y + 1, w, h,
                img, 0, 0, wx.COPY, True)

        img.SelectObject(wx.NullBitmap)

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateGrid()

        self.SetSize((700, 450))
        self.SetTitle('Custom cell renderer')
        self.Centre()

    def CreateGrid(self):

        grid = wx.grid.Grid(self)
        grid.CreateGrid(10, 10)

        img = wx.Bitmap("hightatras.jpg",
                        wx.BITMAP_TYPE_JPEG)

```

```

imgRenderer = MyImageRenderer(img)

grid.SetCellRenderer(0, 0, imgRenderer)
grid.SetColSize(0, img.GetWidth() + 2)
grid.SetRowSize(0, img.GetHeight() + 2)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We create a custom cell renderer for drawing an image in the grid cell.

```

class MyImageRenderer(wx.grid.GridCellRenderer):

    def __init__(self, img, *args, **kw):
        super(MyImageRenderer, self).__init__(*args, **kw)

        self.img = img

```

A custom cell renderer derives from the `wx.grid.GridCellRenderer` class. We store the image in the `img` variable.

```

def Draw(self, grid, attr, dc, rect,
        row, col, isSelected):

    img = wx.MemoryDC()
    img.SelectObject(self.img)

    w, h = self.img.GetWidth(), self.img.GetHeight()

    dc.Blit(rect.x + 1, rect.y + 1, w, h,
            img, 0, 0, wx.COPY, True)

    img.SelectObject(wx.NullBitmap)

```

The actual rendering of a cell is done inside the `Draw` method. One of the parameters of the method is the device context, on which we can perform our drawing. We use the `Blit` method to copy the image to the cell.

```

img = wx.Bitmap("hightatras.jpg",
                wx.BITMAP_TYPE_JPEG)

```

This is the bitmap that we show in the cell.

```

imgRenderer = MyImageRenderer(img)

```

Our custom cell renderer is created; it takes the bitmap as a parameter.

```

grid.SetCellRenderer(0, 0, imgRenderer)
grid.SetColSize(0, img.GetWidth() + 2)
grid.SetRowSize(0, img.GetHeight() + 2)

```


We set the custom cell renderer for the top-left cell of the grid. We update the size of the cell to fit the image size.

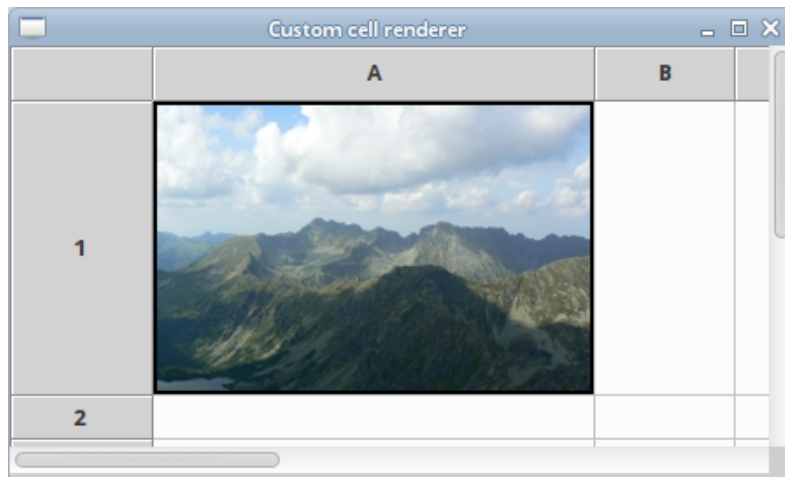


Figure 7.5: Custom cell renderer

Figure 7.5 shows an image drawn in the top-left cell of the grid.

7.11 Model & view

Model & view is a well known software design pattern. In this pattern, the data is separated from its presentation. The data is handled by the model object and the presentation by the view object. This leads to better maintenance and productivity.

`wx.grid.Grid` supports this pattern. The `wx.grid.PyGridTableBase` class serves as a data model and the `wx.grid.Grid` as the view.

Listing 7.11: Model & view

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this code example, we work with wx.grid.GridTableBase
which serves as a model for wx.grid.Grid.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.grid

class GridModel(wx.grid.GridTableBase):

    def __init__(self, *args, **kw):
```

```

        super(GridModel, self).__init__(*args, **kw)

        self.colLabels = ['Name', 'Phone number']

        self.data = [['Jane', '0812 211 222'],
                      ['Thomas', '0812 452 788'],
                      ['Lucy', '0813 456 876'],
                      ['Robert', '0814 675 234']]

    def GetNumberRows(self):

        return len(self.data)

    def GetNumberCols(self):

        return len(self.data[0])

    def GetValue(self, row, col):

        try:
            return self.data[row][col]

        except IndexError:
            return ''

    def GetColLabelValue(self, col):

        return self.colLabels[col]

class MyGrid(wx.grid.Grid):

    def __init__(self, *args, **kw):
        super(MyGrid, self).__init__(*args, **kw)

        model = GridModel()
        self.SetTable(model, True)
        self.AutoSizeColumns(False)

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateGrid()

        self.SetSize((700, 450))
        self.SetTitle('Model & view')
        self.Centre()

    def CreateGrid(self):

```

```

        grid = MyGrid(self)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

We create a simple example with a model and view pattern. There are two columns and four rows. The data in the cells is readonly.

```

class GridModel(wx.grid.GridTableBase):

    def __init__(self, *args, **kw):
        super(GridModel, self).__init__(*args, **kw)
    ...

```

To create a data model, we derive from the `wx.grid.GridTableBase`.

```

self.colLabels = ['Name', 'Phone number']

```

Here we store the column labels. They will be used by the `wx.grid.Grid!GetColLabelValue` method.

```

self.data = [['Jane', '0812 211 222'],
              ['Thomas', '0812 452 788'],
              ['Lucy', '0813 456 876'],
              ['Robert', '0814 675 234']]

```

In the `data` list, we store our data. The `GetValue` method will use this data to display it in the grid cells.

```

def GetNumberRows(self):

    return len(self.data)

def GetNumberCols(self):

    return len(self.data[0])

```

These two methods return the number of rows and columns in the grid. They must be implemented in order to see any cells.

```

def GetValue(self, row, col):

    try:
        return self.data[row][col]

    except IndexError:
        return ''

```

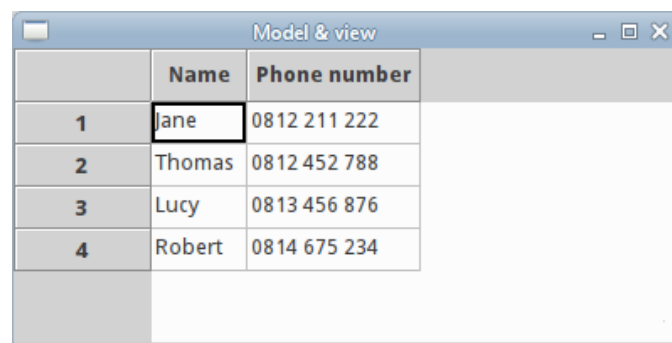
The `GetValue` method is used to get the data for a specific cell from the data source.

```
def GetColLabelValue(self, col):  
    return self.colLabels[col]
```

The `GetColLabelValue` method returns data that is going to be displayed in the column headers.

```
class MyGrid(wx.grid.Grid):  
    def __init__(self, *args, **kw):  
        super(MyGrid, self).__init__(*args, **kw)  
  
        model = GridModel()  
        self.SetTable(model, True)  
        self.AutoSizeColumns(False)
```

This is the view object. We use the `SetTable` method to associate a model with the view object.



	Name	Phone number
1	Jane	0812 211 222
2	Thomas	0812 452 788
3	Lucy	0813 456 876
4	Robert	0814 675 234

Figure 7.6: Model & view

Figure 7.6 shows a `wx.grid.Grid` which was built using the Model & view design pattern.

7.12 Model & view II

In this section, we enable the modification of the data in the grid.

```
$ cat data.csv  
Jane,0812 211 222  
Thomas,0812 452 788  
Lucy,0813 456 876  
Robert,0814 675 234
```

Our data source will be a CSV file.

Listing 7.12: Model & view II

```
#!/usr/bin/env python
```

```

"""
ZetCode Advanced wxPython tutorial

In the code example, we enable data modification
and we export the data in a CSV file.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.grid
import csv

class GridModel(wx.grid.GridTableBase):

    def __init__(self, data):
        super(GridModel, self).__init__()

        self.colLabels = ['Name', 'Phone number']

        self.data = data

    def GetNumberRows(self):

        return len(self.data)

    def GetNumberCols(self):

        return len(self.data[0])

    def GetValue(self, row, col):

        try:
            return self.data[row][col]

        except IndexError:
            return ''

    def SetValue(self, row, col, value):

        try:
            self.data[row][col] = value

        except IndexError:
            return False

        return

    def GetColLabelValue(self, col):

        return self.colLabels[col]

```

```

class MyGrid(wx.grid.Grid):

    def __init__(self, *args, **kw):
        super(MyGrid, self).__init__(*args, **kw)

        self.ReadData()

        model = GridModel(self.data)
        self.SetTable(model, True)
        self.AutoSizeColumns(False)

        self.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)

    def ReadData(self):

        self.data = []

        rd = csv.reader(open('data.csv', 'r'),
                        delimiter=',')

        for row in rd:
            self.data.append(row)

    def OnKeyDown(self, e):

        key = e.GetKeyCode()

        if e.ControlDown() and key == ord('S'):

            wr = csv.writer(open('data.csv', 'w'),
                            delimiter=',')

            for row in self.data:
                wr.writerow(row)

            e.Skip()

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.CreateGrid()

        self.SetSize((700, 450))
        self.SetTitle('Grid')
        self.Centre()

    def CreateGrid(self):

        grid = MyGrid(self)

```

```
def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()
```

The data is read from the data.csv file. We can modify the cells. The Ctrl+S keyboard shortcut will save the changes in the data.csv file.

```
import csv
```

We will use the csv Python module for working with files in a csv (Comma Separated Values) format.

```
def SetValue(self, row, col, value):

    try:
        self.data[row][col] = value

    except IndexError:
        return False

    return
```

The implementation of the `SetValue` enables the modification of the values in the cells.

```
def ReadData(self):

    self.data = []

    rd = csv.reader(open('data.csv', 'rb'),
                    delimiter=',')

    for row in rd:
        self.data.append(row)
```

The `ReadData` method is used to read the values from the data.csv file and fill the `data` list. This list is queried by the `GetValue` method of the `wx.grid.GridTableBase` to display the data in the grid.

```
model = GridModel(self.data)
```

The data list is passed to the model.

```
def OnKeyDown(self, e):

    key = e.GetKeyCode()

    if e.ControlDown() and key == ord('S'):

        wr = csv.writer(open('data.csv', 'wb'),
                        delimiter=',')
```

```
        for row in self.data:
            wr.writerow(row)

    e.Skip()
```

The Control+S keyboard shortcut saves the modified data in the data.csv file.

Chapter 8

wx.richtext.RichTextCtrl

The `wx.richtext.RichTextCtrl` is an enhanced text control. It is capable of displaying various text styles, symbols, images, text paragraphs and hyperlinks. It has a built-in undo/redo functionality.

8.1 Introductory example

In the first example, we set up a simple rich text control and introduce a few methods.

Listing 8.1: Introductory example

```
def InitUI(self):

    self.rt = rtc.RichTextCtrl(self)
    txt = "Python Ruby PHP Tcl Perl\nC++ Java C C# Vala"
    self.rt.AppendText(txt)

    print(self.rt.GetNumberOfLines())
    print(self.rt.GetValue())

    self.SetSize((750, 400))
    self.SetTitle('RichTextCtrl')
    self.Centre()
```

We create a `wx.richtext.RichTextCtrl`. We append two lines to the control and print the returned values of two rich text control methods to the console.

```
self.rt = rtc.RichTextCtrl(self)
```

A rich text control is created.

```
txt = "Python Ruby PHP Tcl Perl\nC++ Java C C# Vala"
self.rt.AppendText(txt)
```

Two lines are appended to the rich text control with the `AppendText` method.

```
print(self.rt.GetNumberOfLines())
```

The `GetNumberOfLines` method returns the number of lines in the rich text control.

```
print(self.rt.GetValue())
```

The `GetValue` method returns the contents of the rich text control as a string.

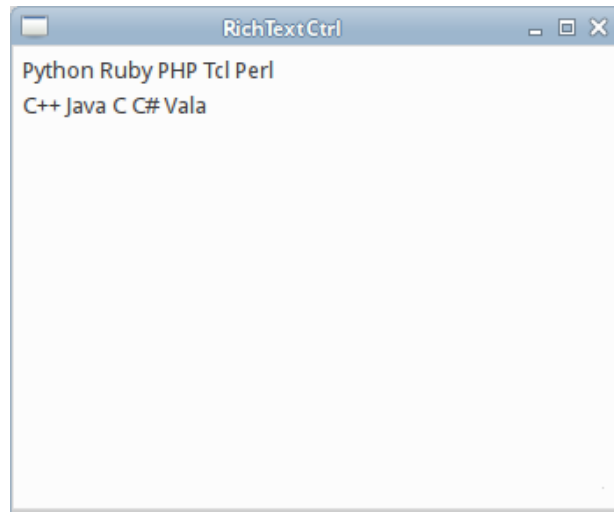


Figure 8.1: Introductory rich text control example

Figure 8.1 shows a `wx.richtext.RichTextCtrl` with two lines.

8.2 Methods

In the second example, we continue explaining rich text control methods. We change the appearance of the text written to the control.

Listing 8.2: Methods

```
def InitUI(self):  
    self.rt = rtc.RichTextCtrl(self)  
    self.TestMethods(self.rt)  
  
    self.SetSize((350, 290))  
    self.SetTitle('RichTextCtrl')  
    self.Centre()  
  
def TestMethods(self, rt):  
    rt.BeginItalic()  
    rt.WriteText("Jane is 17 years old.")  
    rt.EndItalic()  
  
    rt.Newline()  
  
    rt.BeginBold()
```

```

rt.WriteText("I have three apples in the basket.")
rt.EndBold()

rt.Newline()

rt.BeginItalic()
rt.BeginBold()
rt.WriteText("Leo Tolstoy wrote War and Peace.")
rt.EndBold()
rt.EndItalic()

rt.Newline()

rt.BeginFontSize(14)
rt.WriteText("Magnesium is an alkaline earth metal.")
rt.EndFontSize()

rt.Newline()

rt.BeginTextColour(wx.BLUE)
rt.WriteText("There is a cup of tea on the table.")
rt.EndTextColour()

```

In the example, we have some text in italics and bold. We change the font size, add new lines and change the text colour.

```

self.rt = rtc.RichTextCtrl(self)
self.TestMethods(self.rt)

```

An instance of the `wx.richtext.RichTextCtrl` is built. We test the various methods in the `TestMethods` method.

```

rt.BeginItalic()
rt.WriteText("Jane is 17 years old.")
rt.EndItalic()

```

The text between the `BeginItalic` and `EndItalic` methods is displayed in italics. The `WriteText` method writes text to the rich text control.

```

rt.Newline()

```

The `NewLine` method starts a new line in the rich text control.

```

rt.BeginBold()
rt.WriteText("I have three apples in the basket.")
rt.EndBold()

```

We write a sentence in bold with `BeginBold` and `EndBold`.

```

rt.BeginItalic()
rt.BeginBold()
rt.WriteText("Leo Tolstoy wrote War and Peace.")
rt.EndBold()
rt.EndItalic()

```

Various text styles can be nested; here we have a sentence both in italics and bold.

```
rt.BeginFontSize(14)
rt.WriteText("Magnesium is an alkaline earth metal.")
rt.EndFontSize()
```

With `BeginFontSize` and `EndFontSize` the font size is changed.

```
rt.BeginTextColour(wx.BLUE)
rt.WriteText("There is a cup of tea on the table.")
rt.EndTextColour()
```

We change a text colour for a line with `BeginTextColour` and `EndTextColour`.

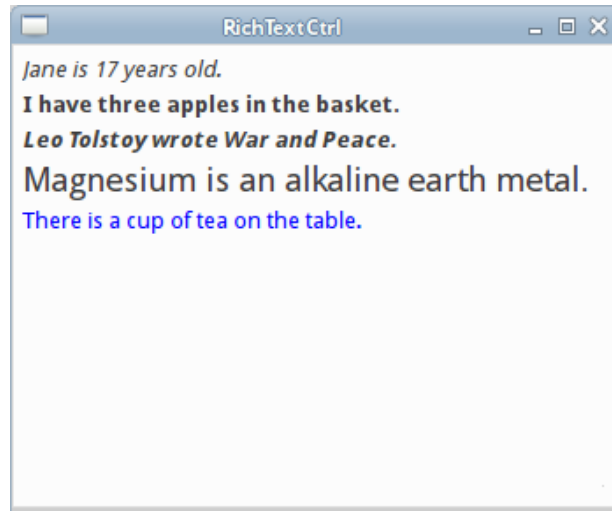


Figure 8.2: `wx.richtext.RichTextCtrl` methods

Figure 8.2 shows a few sentences whose appearance was modified by rich text control methods.

8.3 Bullets

We can have a list of particular items in a text control. In rich text controls, it is a common practice to start each of the items with a particular symbol called a bullet. The bullets are usually letters, numbers, or circles.

Listing 8.3: Bullets

```
def InitUI(self):
    self.rt = rtc.RichTextCtrl(self)
    self.TestBullets(self.rt)

    self.SetSize((350, 290))
    self.SetTitle('Bullets')
    self.Centre()

def TestBullets(self, rt):
```

```

rt.Freeze()
rt.WriteText("Trees:")
rt.Newline()

rt.BeginNumberedBullet(1, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_ROMAN_UPPER)
rt.WriteText("Oak")
rt.Newline()
rt.EndNumberedBullet()

rt.BeginNumberedBullet(2, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_ROMAN_UPPER)
rt.WriteText("Lime")
rt.Newline()
rt.EndNumberedBullet()

rt.BeginNumberedBullet(3, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_ROMAN_UPPER)
rt.WriteText("Pine")
rt.Newline()
rt.EndNumberedBullet()

rt.Newline()

rt.WriteText("Gemstones:")
rt.Newline()

rt.BeginNumberedBullet(1, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_STANDARD )
rt.WriteText("Amethyst")
rt.Newline()
rt.EndNumberedBullet()

rt.BeginNumberedBullet(2, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_STANDARD )
rt.WriteText("Ruby")
rt.Newline()
rt.EndNumberedBullet()

rt.BeginNumberedBullet(3, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_STANDARD )
rt.WriteText("Citrine")
rt.Newline()
rt.EndNumberedBullet()

rt.Newline()
rt.Thaw()

```

In our example, we have two lists; one of them uses upper roman numbers and the other uses standard circles.

```

rt.Freeze()
...
rt.Thaw()

```

For much faster and smoother processing, we put the methods between the `Freeze` and `Thaw` methods.

```

rt.BeginNumberedBullet(1, 100, 60,

```

```

        bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_ROMAN_UPPER)
rt.WriteText("Oak")
rt.Newline()
rt.EndNumberedBullet()

```

These lines create one list item. The bullet is a roman I. The parameters of the `BeginNumberedBullet` method are the bullet number to be shown, the left indent and subindent, and the bullet style.

```

rt.BeginNumberedBullet(1, 100, 60,
    bulletStyle=wx.TEXT_ATTR_BULLET_STYLE_STANDARD )
rt.WriteText("Amethyst")
rt.Newline()
rt.EndNumberedBullet()

```

This is the first item of the gemstones list. The bullets are small circles.

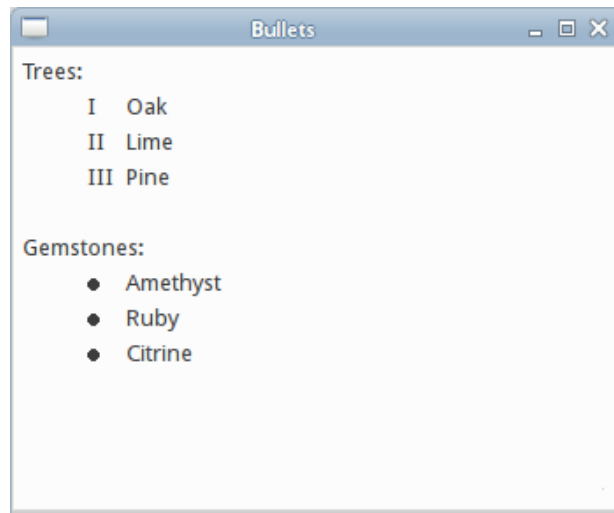


Figure 8.3: Bullets

Figure 8.3 shows roman style and standard circular bullets in the rich text control.

8.4 Images and URL links

A `wx.richtext.RichTextCtrl` can display images. We can also construct URL links.

Listing 8.4: Image and URL link

```

#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this example, we show an image
and a URL link in the wx.richtext.RichTextCtrl.

```

Author: Jan Bodnar
Website: zetcode.com
"""

```
import wx
import wx.richtext as rtc
import webbrowser
```

```
class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.rt = rtc.RichTextCtrl(self)
        self.rt.Bind(wx.EVT_TEXT_URL, self.OnClickUrl)
        self.AddObjects(self.rt)

        self.SetSize((750, 400))
        self.SetTitle('Image and URL')
        self.Centre()

    def AddObjects(self, rt):

        rt.AppendText("The High Tatras")
        rt.Newline()
        rt.Newline()
        rt.WriteImage("hightatras.jpg",
            bitmapType=wx.BITMAP_TYPE_JPEG)
        rt.Newline()
        rt.Newline()

        rt.BeginUnderline()
        rt.BeginTextColour((0, 0, 255))
        url = "http://en.wikipedia.org/wiki/High_Tatras"
        rt.BeginURL(url)
        rt.WriteText("High Tatras")
        rt.EndURL()
        rt.EndTextColour()
        rt.EndUnderline()

        rt.Newline()
        rt.Newline()

    def OnClickUrl(self, e):

        webbrowser.open(e.GetString())

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
```

```
app.MainLoop()
```

```
if __name__ == "__main__":  
    main()
```

An image is shown in the rich text control. Below the image, we construct a URL link.

```
import webbrowser
```

We use the `webbrowser` Python module to open a link in the default web browser.

```
self.rt.Bind(wx.EVT_TEXT_URL, self.OnClickUrl)
```

When we click on the URL link, the `OnClickUrl` method is called.

```
rt.WriteImageFile("hightatras.jpg",  
    bitmapType=wx.BITMAP_TYPE_JPEG)
```

The `WriteImageFile` method writes an image to the rich text control.

```
rt.BeginUnderline()  
rt.BeginTextColour((0, 0, 255))  
url = "http://en.wikipedia.org/wiki/High_Tatras"  
rt.BeginURL(url)  
rt.WriteText("High Tatras")  
rt.EndURL()  
rt.EndTextColour()  
rt.EndUnderline()
```

A URL link is created with `BeginURL` and `EndURL`. The URL text is underlined and written in blue colour, which is how URL links are usually styled.

```
def OnClickUrl(self, e):  
    webbrowser.open(e.GetString())
```

Upon clicking on the URL link, a web page is opened in the default web browser.

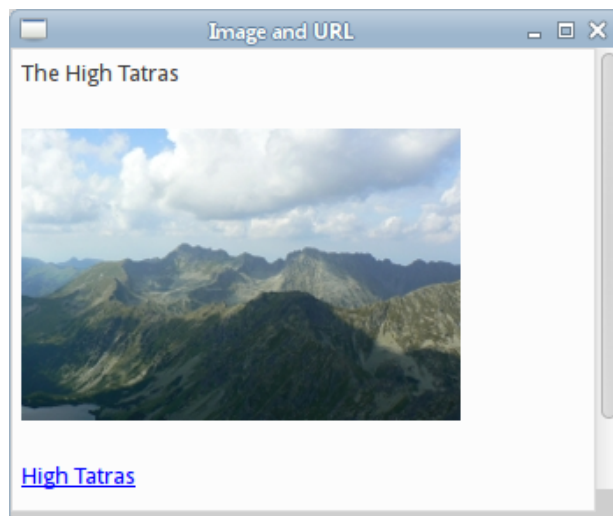


Figure 8.4: Image and URL link

Figure 8.4 shows an image and a URL link in the `wx.richtext.RichTextCtrl`.

8.5 Font weight and style

In the next example, we change the font weight and style.

Listing 8.5: Font weight & style

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

In this code example we create toolbar buttons
to change the font weight and font style of the text.

Author: Jan Bodnar
Website: zetcode.com
"""

import wx
import wx.richtext as rtc

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.BuildToolBar()
        self.rt = rtc.RichTextCtrl(self)
```

```

        self.SetSize((750, 400))
        self.SetTitle('RichTextCtrl')
        self.Centre()

def BuildToolBar(self):

    tlb = self.CreateToolBar()
    btext = tlb.AddCheckTool(wx.ID_ANY, 'Bold',
                             wx.Bitmap('bold.png'))
    itext = tlb.AddCheckTool(wx.ID_ANY, 'Italic',
                             wx.Bitmap('italic.png'))
    utext = tlb.AddCheckTool(wx.ID_ANY, 'Underline',
                             wx.Bitmap('underline.png'))

    tlb.Realize()

    self.Bind(wx.EVT_TOOL, self.OnBoldSelected,
              btext)
    self.Bind(wx.EVT_TOOL, self.OnItalSelected,
              itext)
    self.Bind(wx.EVT_TOOL, self.OnUnderSelected,
              utext)

    self.Bind(wx.EVT_UPDATE_UI, self.OnUpdateBold,
              btext)
    self.Bind(wx.EVT_UPDATE_UI, self.OnUpdateItalic,
              itext)
    self.Bind(wx.EVT_UPDATE_UI, self.OnUpdateUnderline,
              utext)

def OnBoldSelected(self, e):
    self.rt.ApplyBoldToSelection()

def OnItalSelected(self, e):
    self.rt.ApplyItalicToSelection()

def OnUnderSelected(self, e):
    self.rt.ApplyUnderlineToSelection()

def OnUpdateBold(self, e):
    e.Check(self.rt.IsSelectionBold())

def OnUpdateItalic(self, e):
    e.Check(self.rt.IsSelectionItalics())

def OnUpdateUnderline(self, e):
    e.Check(self.rt.IsSelectionUnderlined())

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

There is a toolbar whose tool items make the text bold, italic, and underlined. The check tool items on the toolbar appear pressed or not pressed, depending on the state of the text where the caret is located.

```
tlb = self.CreateToolBar()
btext = tlb.AddCheckLabelTool(wx.ID_ANY, 'Bold',
                               wx.Bitmap('bold.png'))
itext = tlb.AddCheckLabelTool(wx.ID_ANY, 'Italic',
                               wx.Bitmap('italic.png'))
...
```

We create a toolbar and add check label tool items. Each item is represented by a bitmap. These tool items have two basic states: pressed and not pressed.

```
self.Bind(wx.EVT_TOOL, self.OnBoldSelected,
           btext)
self.Bind(wx.EVT_TOOL, self.OnItalSelected,
           itext)
...
```

Here the tool items are bound to their methods. For example, if we press the bold tool item, the `OnBoldSelected` method is launched

```
self.Bind(wx.EVT_UPDATE_UI, self.OnUpdateBold,
           btext)
self.Bind(wx.EVT_UPDATE_UI, self.OnUpdateItalic,
           itext)
...
```

The `wx.EVT_UPDATE_UI` event is triggered, when the state of the tool items might change.

```
def OnBoldSelected(self, e):
    self.rt.ApplyBoldToSelection()
```

The `ApplyBoldToSelection` method of the rich text control is called to make the selected text bold.

```
def OnUpdateBold(self, e):
    e.Check(self.rt.IsSelectionBold())
```

In the `OnUpdateBold` method, we check if the selected text is bold using the `IsSelectionBold` method. If it is bold, the corresponding toolbar item is checked. If not, it is unchecked.

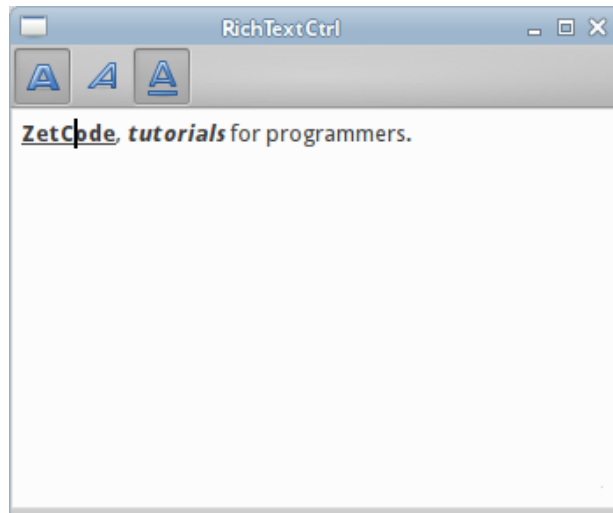


Figure 8.5: Font weight & style

Figure 8.5 shows text in the rich text control with different font weight and style. The state of the toolbar items reflect the text appearance.

8.6 Undo and redo

Undo and redo functionality is built in the `wx.richtext.RichTextCtrl`. In the next example we show how to use it.

Listing 8.6: Undo, redo

```
def InitUI(self):

    self.BuildToolBar()

    self.rt = rtc.RichTextCtrl(self)

    self.SetSize((350, 290))
    self.SetTitle('Undo, redo')
    self.Centre()

def BuildToolBar(self):

    tlb = self.CreateToolBar()
    tundo = tlb.AddLabelTool(wx.ID_UNDO, 'Undo',
                             wx.Bitmap('undo.png'))
    tredo = tlb.AddLabelTool(wx.ID_REDO, 'Redo',
                              wx.Bitmap('redo.png'))
    tlb.Realize()

    self.Bind(wx.EVT_TOOL, self.ForwardEvent, tundo)
    self.Bind(wx.EVT_TOOL, self.ForwardEvent, tredo)
    self.Bind(wx.EVT_UPDATE_UI, self.ForwardEvent, tundo)
    self.Bind(wx.EVT_UPDATE_UI, self.ForwardEvent, tredo)
```

```
def ForwardEvent(self, e):
    self.rt.ProcessEvent(e)
```

There is a toolbar with undo and redo tool items. They undo and redo the changes made in the rich text control text.

```
tlb = self.CreateToolBar()
tundo = tlb.AddLabelTool(wx.ID_UNDO, 'Undo',
    wx.Bitmap('undo.png'))
tredo = tlb.AddLabelTool(wx.ID_REDO, 'Redo',
    wx.Bitmap('redo.png'))
tlb.Realize()
```

The toolbar with the tool items is created. The tool items must have the `wx.ID_UNDO` and `wx.ID_REDO` ids.

```
self.Bind(wx.EVT_TOOL, self.ForwardEvent, tundo)
self.Bind(wx.EVT_TOOL, self.ForwardEvent, tredo)
self.Bind(wx.EVT_UPDATE_UI, self.ForwardEvent, tundo)
self.Bind(wx.EVT_UPDATE_UI, self.ForwardEvent, tredo)
```

We bind the `wx.EVT_TOOL` and the `wx.EVT_UPDATE_UI` events. They both launch the `ForwardEvent` method.

```
def ForwardEvent(self, e):
    self.rt.ProcessEvent(e)
```

The undo and redo functionality is achieved by simply forwarding the events to the `ProcessEvent` method of the rich text control. Remember that we must choose the standard ids for the tool items.

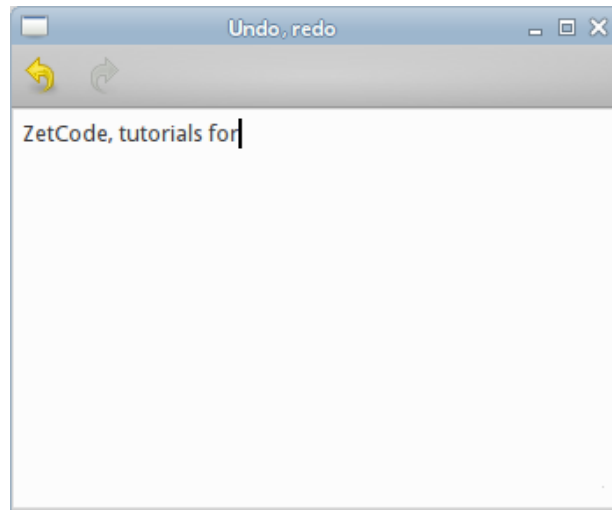


Figure 8.6: Undo, redo

Figure 8.6 shows text in the rich text control. The first tool item signals that the text can be undone.

8.7 Saving in XML

The contents of a `wx.richtext.RichTextCtrl` can be saved in three different formats: textual, HTML, or XML. If a rich text is saved in a txt format, the formatting of the text is lost; only the text is saved. In case of an XML format, the formatting is preserved. An XML file also has instructions for formatting the text.

In order to be able to save or load files in a specific format, we need to activate its handler. The `wx.richtext.RichTextXMLHandler` is a handler for XML files. In the following example, we save data in an XML file.

Listing 8.7: Saving in XML

```
#!/usr/bin/env python

'''
ZetCode Advanced wxPython tutorial

In this code example we save the contents
of the wx.richtext.RichTextCtrl in an XML file.

Author: Jan Bodnar
Website: zetcode.com
'''

import wx
import wx.richtext as rtc

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)

        self.InitUI()

    def InitUI(self):

        self.rt = rtc.RichTextCtrl(self)

        xh = wx.richtext.RichTextXMLHandler()
        wx.richtext.RichTextBuffer.AddHandler(xh)

        self.rt.SetFilename('test.xml')

        self.WriteRichText(self.rt)

        self.CreateShortcut()

        self.SetSize((750, 400))
        self.SetTitle('Save in XML')
        self.Centre()

    def WriteRichText(self, rt):

        rt.BeginItalic()
        rt.WriteText("Jane is 17 years old.")
```

```

        rt.EndItalic()

        rt.Newline()

        rt.BeginItalic()
        rt.BeginBold()
        rt.WriteText("Leo Tolstoy wrote War and Peace.")
        rt.EndBold()
        rt.EndItalic()

def CreateShortcut(self):

    nid = wx.NewIdRef()

    self.Bind(wx.EVT_MENU, self.OnSaveXML, id=nid)

    accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,
        ord('S'), nid)])
    self.SetAcceleratorTable(accel_tbl)

def OnSaveXML(self, e):

    self.rt.SaveFile('', wx.richtext.RICHTEXT_TYPE_XML)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

Two formatted sentences are created. The Ctrl+S keyboard shortcut saves the text in the test.xml file.

```
self.rt = rtc.RichTextCtrl(self)
```

The rich text control is created.

```
xh = wx.richtext.RichTextXMLHandler()
wx.richtext.RichTextBuffer.AddHandler(xh)
```

The XML handler is activated. Now we can save and load XML files.

```
self.rt.SetFilename('test.xml')
```

We set the current file to be the test.xml file with `SetFilename`

```
def WriteRichText(self, rt):

    rt.BeginItalic()
    rt.WriteText("Jane is 17 years old.")
    rt.EndItalic()
...

```

In the `WriteRichText` method, we write two sentences to the rich text control.

```
def CreateShortcut(self):  
  
    nid = wx.NewIdRef()  
  
    self.Bind(wx.EVT_MENU, self.OnSaveXML, id=nid)  
  
    accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,  
                                     ord('S'), nid)])  
    self.SetAcceleratorTable(accel_tbl)
```

The `CreateShortcut` method creates a Ctrl+S keyboard shortcut. It calls the `OnSaveXML` method, which will save the contents of the rich text control to the disk.

```
def OnSaveXML(self, e):  
  
    self.rt.SaveFile('', wx.richtext.RICHTEXT_TYPE_XML)
```

We use the `SaveFile` method to save the contents of the rich text control. The first parameter of the method is the file name. If it is empty, we are going to write to the current file. We have specified the current file with the `SetFilename` method. The second parameter is the file type. For XML files, we use the `wx.richtext.RICHTEXT_TYPE_XML`.

8.8 Loading from XML

In the previous example, we have saved the contents of a rich text control in an XML file. Now we are going to perform the reverse operation. We load data from the XML file and display it in the rich text control.

Listing 8.8: Loading from XML

```
#!/usr/bin/env python  
  
"""  
ZetCode Advanced wxPython tutorial  
  
In this code example we load an xml file  
into the wx.richtext.RichTextCtrl.  
  
Author: Jan Bodnar  
Website: zetcode.com  
"""  
  
import wx  
import wx.richtext as rtc  
  
class Example(wx.Frame):  
  
    def __init__(self, *args, **kw):  
        super(Example, self).__init__(*args, **kw)  
  
        self.InitUI()
```



```

def InitUI(self):

    self.rt = rtc.RichTextCtrl(self)
    xh = wx.richtext.RichTextXMLHandler()
    wx.richtext.RichTextBuffer.AddHandler(xh)

    self.CreateShortcut()

    self.SetSize((750, 400))
    self.SetTitle('Load from XML')
    self.Centre()

def CreateShortcut(self):

    nid = wx.NewIdRef()

    self.Bind(wx.EVT_MENU, self.LoadXMLFile, id=nid)

    accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,
        ord('L'), nid)])
    self.SetAcceleratorTable(accel_tbl)

def LoadXMLFile(self, e):

    self.rt.LoadFile('test.xml',
        wx.richtext.RICHTEXT_TYPE_XML)

def main():

    app = wx.App()
    ex = Example(None)
    ex.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

The Ctrl+L keyboard shortcut load the test.xml file into the rich text control.

```

self.rt = rtc.RichTextCtrl(self)
xh = wx.richtext.RichTextXMLHandler()
wx.richtext.RichTextBuffer.AddHandler(xh)

```

A rich text control is created and the XML handler is activated.

```

def CreateShortcut(self):

    nid = wx.NewIdRef()

    self.Bind(wx.EVT_MENU, self.LoadXMLFile, id=nid)

    accel_tbl = wx.AcceleratorTable([(wx.ACCEL_CTRL,
        ord('L'), nid)])
    self.SetAcceleratorTable(accel_tbl)

```

The Ctrl+L keyboard shortcut is created. It calls the LoadXMLFile method.

```
def LoadXMLFile(self, e):  
  
    self.rt.LoadFile('test.xml',  
                     wx.richtext.RICHTEXT_TYPE_XML)
```

In the `LoadXMLFile` method, we call the `LoadFile` method. The first parameter is the file name to load; the second parameter is the file type.

Part III

Games

Chapter 9

Snake

Snake is an older classic video game. It was first created in late 70s. Later it was brought to PCs. In this game the player controls a snake. The objective is to eat as many apples as possible. Each time the snake eats an apple, its body grows. The snake must avoid the walls and itself.

Listing 9.1: Snake game

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

This is a simple Snake game clone.

Author: Jan Bodnar
Website: zetcode.com
"""

import random
import sys
import wx

class n:
    """Stores constants and variables to avoid
    global values"""

    WIDTH = 300
    HEIGHT = 300
    DOT_SIZE = 10
    ALL_DOTS = WIDTH * HEIGHT // (DOT_SIZE * DOT_SIZE)
    RAND_POS = 29
    DELAY = 140
    TIMER_ID = 1

    x = [0] * ALL_DOTS
    y = [0] * ALL_DOTS

class Board(wx.Panel):

    def __init__(self, parent):
```

```

super(Board, self).__init__(parent,
                             size=(n.WIDTH, n.HEIGHT), style=wx.WANTS_CHARS)

self.SetFocus()
self.InitGame()

self.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
self.Bind(wx.EVT_PAINT, self.OnPaint)
self.Bind(wx.EVT_TIMER, self.OnTimer, id=n.TIMER_ID)

def InitGame(self):
    """Initiates variables, loads images, places the
       apple randomly and starts the game cycle."""

    self.SetDoubleBuffered(True)

    self.InitiateVariables()
    self.LoadImages()
    self.PlaceAppleRandomly()
    self.StartGameCycle()

def StartGameCycle(self):
    """Creates and starts a timer, which is used
       to build a game cycle."""

    self.timer = wx.Timer(self, n.TIMER_ID)
    self.timer.Start(n.DELAY)

def InitiateVariables(self):
    """Initiates important variables"""

    self.left = False
    self.right = True
    self.up = False
    self.down = False
    self.inGame = True
    self.dots = 3

    for i in range(self.dots):
        n.x[i] = 50 - i * 10
        n.y[i] = 50

def LoadImages(self):
    """Loads game sprites."""

    try:
        self.ball = wx.Bitmap("dot.png")
        self.apple = wx.Bitmap("apple.png")
        self.head = wx.Bitmap("head.png")

    except Exception as e:

        print(e.message)
        sys.exit(1)

def OnTimer(self, e):
    """Creates a game cycle."""

```

```

        if self.inGame:
            self.CheckCollisionWithApple()
            self.CheckCollisions()
            self.MoveSnake()
        else:
            self.timer.Stop()

        self.Refresh()

def OnPaint(self, e):
    """Handles paint events"""

    dc = wx.PaintDC(self)

    self.DrawBoardBackGround(dc)

    if self.inGame:
        self.DrawGameObjects(dc)
    else:
        self.GameOver(dc)

def DrawBoardBackGround(self, dc):
    """Paints the background of the panel in
    black color"""

    dc.SetBrush(wx.Brush('#000000'))
    w, h = self.GetClientSize()
    dc.DrawRectangle(0, 0, w, h)

def DrawGameObjects(self, dc):
    """Draws apple and snake images on the board"""

    dc.DrawBitmap(self.apple, self.apple_x, self.apple_y)

    for z in range(self.dots):
        if z == 0:
            dc.DrawBitmap(self.head, n.x[z], n.y[z])
        else:
            dc.DrawBitmap(self.ball, n.x[z], n.y[z])

def GameOver(self, dc):
    """Draws game over on the board"""

    msg = "Game Over"
    small = wx.Font(12, wx.FONTFAMILY_DEFAULT,
        wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_BOLD)
    dc.SetFont(small)

    w, h = dc.GetTextExtent(msg)
    cw, ch = self.GetClientSize()
    dc.SetTextForeground("#ffffff")
    dc.SetDeviceOrigin(cw//2, ch//2)

    dc.DrawText(msg, -w//2, 0)

def CheckCollisionWithApple(self):

```

```

        """Checks if the snake collides with the apple."""

        if n.x[0] == self.apple_x and n.y[0] == self.apple_y:
            self.dots = self.dots + 1
            self.PlaceAppleRandomly()

def MoveSnake(self):
    """Moves the snake"""

    z = self.dots

    while z > 0:
        n.x[z] = n.x[(z - 1)]
        n.y[z] = n.y[(z - 1)]
        z = z - 1

    if self.left:
        n.x[0] -= n.DOT_SIZE

    if self.right:
        n.x[0] += n.DOT_SIZE

    if self.up:
        n.y[0] -= n.DOT_SIZE

    if self.down:
        n.y[0] += n.DOT_SIZE

def CheckCollisions(self):
    """Checks collisions leading to game over."""

    z = self.dots

    while z > 0:
        if z > 4 and n.x[0] == n.x[z] \
            and n.y[0] == n.y[z]:

            self.inGame = False

            z = z - 1

    if n.y[0] >= n.HEIGHT:
        self.inGame = False

    if n.y[0] < 0:
        self.inGame = False

    if n.x[0] >= n.WIDTH:
        self.inGame = False

    if n.x[0] < 0:
        self.inGame = False

def PlaceAppleRandomly(self):
    """Places the apple randomly on the board within
    the size of the board."""

    r = random.randint(0, n.RAND_POS)
    self.apple_x = r * n.DOT_SIZE

```

```

        r = random.randint(0, n.RAND_POS)
        self.apple_y = r * n.DOT_SIZE

    def OnKeyDown(self, e):
        """Reacts to cursor key events."""

        key = e.GetKeyCode()

        if key == wx.WXK_LEFT and not self.right:
            self.left = True
            self.up = False
            self.down = False

        if key == wx.WXK_RIGHT and not self.left:
            self.right = True
            self.up = False
            self.down = False

        if key == wx.WXK_UP and not self.down:
            self.up = True
            self.right = False
            self.left = False

        if key == wx.WXK_DOWN and not self.up:
            self.down = True
            self.right = False
            self.left = False

class Snake(wx.Frame):

    def __init__(self, *args, **kw):
        super(Snake, self).__init__(*args, **kw)

        self.InitUI()
        self.SetTitle("Snake")
        self.Centre()

    def InitUI(self):

        self.board = Board(self)
        self.SetMinClientSize(self.board.GetBestSize())
        self.SetMaxClientSize(self.board.GetBestSize())

def main():

    app = wx.App()
    snake = Snake(None)
    snake.Show()
    app.MainLoop()

if __name__ == '__main__':
    main()

```

The size of each of the joints of a snake is 10 px. The snake is controlled with the cursor keys. Initially, the snake has three joints. The game starts immediately. When the game is finished, we display "Game Over" message in the center of

the window.

```
class n:
    """Stores constants and variables to avoid
       global values"""

    WIDTH = 300
    HEIGHT = 300
    ...
```

The n class is used to store constants and variables to avoid using global variables and constants.

```
WIDTH = 300
HEIGHT = 300
DOT_SIZE = 10
ALL_DOTS = WIDTH * HEIGHT // (DOT_SIZE * DOT_SIZE)
RAND_POS = 29
DELAY = 140
TIMER_ID = 1
```

The WIDTH and HEIGHT constants determine the size of the Board. The DOT_SIZE is the size of the apple and the dot of the snake. The ALL_DOTS constant defines the maximum number of possible dots on the Board. The RAND_POS constant is used to calculate a random position of an apple. The DELAY constant determines the speed of the game. The TIMER_ID stores the Id of a game loop timer.

```
x = [0] * ALL_DOTS
y = [0] * ALL_DOTS
```

These two lists store x and y coordinates of all possible joints of a snake.

```
self.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
self.Bind(wx.EVT_PAINT, self.OnPaint)
self.Bind(wx.EVT_TIMER, self.OnTimer, id=n.TIMER_ID)
```

We have three event handlers. Binding the wx.EVT_KEY_DOWN event we capture key press events. The wx.EVT_PAINT is for doing painting on the board panel. The wx.EVT_TIMER is for creating the game cycle with a timer object.

```
def InitGame(self):
    """Initiates variables, loads images, places the
       apple randomly and starts the game cycle."""

    self.SetDoubleBuffered(True)

    self.InitiateVariables()
    self.LoadImages()
    self.PlaceAppleRandomly()
    self.StartGameCycle()
```

The InitGame method does some preliminary work. It initiates important variables, loads image sprites, places the apple object randomly on the board and starts the game cycle. All these tasks are delegated to specific methods.

```
def InitiateVariables(self):
    '''Initiates important variables'''
```

```

self.left = False
self.right = True
self.up = False
self.down = False
self.inGame = True
self.dots = 3

for i in range(self.dots):
    n.x[i] = 50 - i * 10
    n.y[i] = 50

```

The method initiates game variables. The first four are the direction variables. They control in which direction the snake is moving. At the start of the game, the snake moves to the right. The `inGame` variable controls whether the game is going on. The `dots` variable stores the numbers of joints of the snake. In the for loop, we give the snake joints initial coordinates. It always starts from the same position.

```

def LoadImages(self):
    """Loads game sprites."""

    try:
        self.ball = wx.Bitmap("dot.png")
        self.apple = wx.Bitmap("apple.png")
        self.head = wx.Bitmap("head.png")

    except Exception as e:

        print(e.message)
        sys.exit(1)

```

We load the game images, also called sprites, from the disk. The images are PNG images located in the current working directory. We have three images. One is for the apple, one for the snake joint, and one for the snake head. The head is drawn in red color so that we know where is the beginning of the snake.

```

def OnTimer(self, e):
    """Creates a game cycle."""

    if self.inGame:
        self.CheckCollisionWithApple()
        self.CheckCollisions()
        self.MoveSnake()
    else:
        self.timer.Stop()

    self.Refresh()

```

Every `DELAY` ms, the `OnTimer` method is called. The method builds the game cycle. If we are in the game, we call three methods that build the logic of the game; otherwise we stop the timer.

```

def OnPaint(self, e):
    """Handles paint events"""

    dc = wx.PaintDC(self)

    self.DrawBoardBackGround(dc)

```

```

if self.inGame:
    self.DrawGameObjects(dc)
else:
    self.GameOver(dc)

```

Inside the `OnPaint` method, we a) draw the background of the board panel in black b) check the `inGame` variable. If the variable is true, we draw the game objects; otherwise we draw 'Game Over' string into the center of the board.

```

def DrawGameObjects(self, dc):
    """Draws apple and snake images on the board"""

    dc.DrawBitmap(self.apple, self.apple_x, self.apple_y)

    for z in range(self.dots):
        if z == 0:
            dc.DrawBitmap(self.head, n.x[z], n.y[z])
        else:
            dc.DrawBitmap(self.ball, n.x[z], n.y[z])

```

The `DrawGameObjects` method draws the apple and the joints of the snake. The first joint of a snake is its head, which is represented by a red circle.

```

def GameOver(self, dc):
    """Draws game over on the board"""

    msg = "Game Over"
    small = wx.Font(12, wx.FONTFAMILY_DEFAULT,
                    wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_BOLD)
    dc.SetFont(small)

    w, h = dc.GetTextExtent(msg)
    cw, ch = self.GetClientSize()
    dc.SetTextForeground("#ffffff")
    dc.SetDeviceOrigin(cw//2, ch//2)

    dc.DrawText(msg, -w//2, 0)

```

The `GameOver` method draws the final message on the board. In order to center the message, we need to determine the size of the text with `GetTextExtent`. The client size of the window is retrieved with `GetClientSize`.

```

def CheckCollisionWithApple(self):
    """Checks if the snake collides with the apple."""

    if n.x[0] == self.apple_x and n.y[0] == self.apple_y:
        self.dots = self.dots + 1
        self.PlaceAppleRandomly()

```

In the `CheckCollisionWithApple` method, we check if the snake hits the apple object. We compare the coordinates of the apple and the snake's head. If there is a hit, we increase the number of snake joints and place another apple randomly on the board.

```

def MoveSnake(self):
    """Moves the snake"""

    z = self.dots
    ...

```

The `MoveSnake` method has the key algorithm of the game. To understand it, look at how the snake is moving. You control the head of the snake. You can change its direction with the cursor keys. The rest of the joints move one position up the chain. The second joint moves where the first was, the third joint where the second was etc.

```
while z > 0:
    if z > 4 and n.x[0] == n.x[z] \
        and n.y[0] == n.y[z]:

        self.inGame = False

    z = z - 1
```

In the while loop, we move the joints up the chain.

```
if self.left:
    n.x[0] -= n.DOT_SIZE
```

If the `left` variable is true, we move the head of the snake to the left. This is done by decreasing the x coordinate of the head image by `DOT_SIZE`.

```
def CheckCollisions(self):
    '''Checks collisions leading to game over.'''

    z = self.dots
    ...
```

The `CheckCollisions` method checks if the snake has hit itself or one of the walls. All these collisions end the game.

```
while z > 0:
    if z > 4 and n.x[0] == n.x[z] and n.y[0] == n.y[z]:
        self.inGame = False
    z = z - 1
```

In this while loop, we check if the snake hits one of its joints with the head.

```
if n.y[0] >= n.HEIGHT:
    self.inGame = False
```

If we hit the bottom of the board, the game ends.

```
def PlaceAppleRandomly(self):
    '''Places the apple randomly on the board within
    the size of the board.'''

    r = random.randint(0, n.RAND_POS)
    self.apple_x = r * n.DOT_SIZE
    r = random.randint(0, n.RAND_POS)
    self.apple_y = r * n.DOT_SIZE
```

This method positions a new apple object on the board. We get a random number from 0 to `RAND_POS` and set the x and y coordinates of the apple object.

```
def OnKeyDown(self, e):
    '''Reacts to cursor key events.'''
```

```

key = e.GetKeyCode()

if key == wx.WXK_LEFT and not self.right:
    self.left = True
    self.up = False
    self.down = False
...

```

In the `OnKeyDown` method, we react to key press events. The game is controlled with cursor keys. We set the direction variables. They are used in the `MoveSnake` method to change the coordinates of the snake. The `not self.right` is there to ensure that we cannot move immediately to the opposite direction. In other words, if we are heading to the left, we cannot move to the right without changing the direction to other two directions.

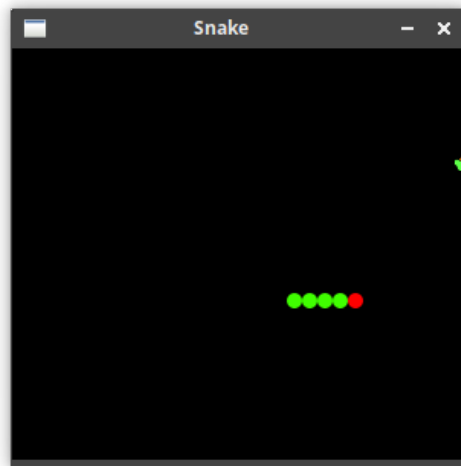


Figure 9.1: Snake game

Figure 9.1 shows a screenshot of the Snake game.

Chapter 10

Sokoban

Sokoban is another classic computer game. It was created in 1980 by Hiroyuki Imabayashi. Sokoban means a warehouse keeper in Japanese. The player pushes boxes around a maze. The objective is to place all boxes in designated locations.

We control the sokoban object with cursor keys. We can also press the R key to restart the level. When all bags are placed on the destination areas, the game is finished. We draw "Completed" string in the left upper corner of the window.

Listing 10.1: Sokoban game

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

This is a simple Sokoban game clone.

Author: Jan Bodnar
Website: zetcode.com
"""

import sys
import wx

class S:
    """Stores constants to avoid global values"""

    SPACE = 20
    OFFSET = 30

    LEFT_COLLISION = 1
    RIGHT_COLLISION = 2
    TOP_COLLISION = 3
    BOTTOM_COLLISION = 4

    level = """
    #####+
    ##    #+
    ##$   #+
    ####  $##+
    ##    $ $ #+
    """
```

```

#### # ## # #####+
## # ## ##### ..#+
## $ $ ..#+
##### ## # @## ..#+
## #####+
#####+
"""

```

```

class Actor(object):
    """Generic actor class for all game objects"""

    def __init__(self, x, y):
        self._x = x
        self._y = y

    def GetImage(self):
        return self.image

    def x(self):
        return self._x

    def y(self):
        return self._y

    def IsLeftCollision(self, actor):
        if ((self.x() - s.SPACE) == actor.x()) and \
            (self.y() == actor.y()):
            return True
        else:
            return False

    def IsRightCollision(self, actor):
        if ((self.x() + s.SPACE) == actor.x()) and \
            (self.y() == actor.y()):
            return True
        else:
            return False

    def IsTopCollision(self, actor):
        if ((self.y() - s.SPACE) == actor.y()) and \
            (self.x() == actor.x()):
            return True
        else:
            return False

    def IsBottomCollision(self, actor):
        if ((self.y() + s.SPACE) == actor.y()) and \
            (self.x() == actor.x()):
            return True
        else:
            return False

class Baggage(Actor):

    def __init__(self, x, y):
        super(Baggage, self).__init__(x, y)

    try:
        self.image = wx.Bitmap("baggage.png")

```

```

        except Exception as e:
            print(e.message)
            sys.exit(1)

    def Move(self, x, y):
        self._x += x
        self._y += y

class Player(Actor):

    def __init__(self, x, y):
        super(Player, self).__init__(x, y)

        try:
            self.image = wx.Bitmap("sokoban.png")

        except Exception as e:

            print(e.message)
            sys.exit(1)

    def Move(self, x, y):
        self._x += x
        self._y += y

class Wall(Actor):

    def __init__(self, x, y):
        super(Wall, self).__init__(x, y)

        try:
            self.image = wx.Bitmap("wall.png")

        except Exception as e:

            print(e.message)
            sys.exit(1)

class Area(Actor):

    def __init__(self, x, y):
        super(Area, self).__init__(x, y)

        try:
            self.image = wx.Bitmap("area.png")
        except Exception as e:

            print(e.message)
            sys.exit(1)

class Board(wx.Panel):
    """Panel on which the game takes place"""

    def __init__(self, parent):
        super(Board, self).__init__(parent,
                                     style=wx.WANTS_CHARS)

        self.InitUI()

```



```

def InitUI(self):

    self.SetDoubleBuffered(True)
    self.SetFocus()

    self.InitVariables()
    self.InitWorld()

    self.Bind(wx.EVT_PAINT, self.OnPaint)
    self.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
    self.SetSize((self.w, self.h))

def InitVariables(self):
    """Initiates game variables"""

    self.world = []
    self.walls = []
    self.baggs = []
    self.areas = []

    self.w = 0
    self.h = 0

    self.completed = False
    self.soko = None

def InitWorld(self):
    """Sets up the game world"""

    x = s.OFFSET
    y = s.OFFSET

    for i in range(len(s.level)):

        item = s.level[i]

        if item == '+':
            y += s.SPACE
            if self.w < x:
                self.w = x

            x = s.OFFSET
        elif item == '#':
            w = Wall(x, y)
            self.walls.append(w)
            x += s.SPACE
        elif item == '$':
            b = Baggage(x, y)
            self.baggs.append(b)
            x += s.SPACE
        elif item == '.':
            a = Area(x, y)
            self.areas.append(a)
            x += s.SPACE
        elif item == '@':
            self.soko = Player(x, y)
            x += s.SPACE
        elif item == " ":

```

```

        x += s.SPACE

    self.h = y

def BuildWorld(self, dc):
    """Constructs the game world"""

    dc.SetBrush(wx.Brush(wx.Colour(250, 240, 170)))
    dc.SetBrush(wx.Brush(wx.Colour('LIGHT STEEL BLUE')))
    # dc.SetPen(wx.Pen(wx.Colour(250, 240, 170)))
    # dc.SetPen(wx.Pen(wx.Colour(250, 240, 170)))
    w, h = self.GetClientSize()
    dc.DrawRectangle(0, 0, w, h)
    # dc.SetPen(wx.Pen(wx.Colour(0, 0, 0)))

    world = self.walls + self.areas + \
            self.baggs + [self.soko]

    for i in range(len(world)):

        item = world[i]

        if isinstance(item, Player) or \
            isinstance(item, Baggage):
            dc.DrawBitmap(item.GetImage(), item.x()+2,
                           item.y()+2)

        else:
            dc.DrawBitmap(item.GetImage(), item.x(),
                           item.y())

    if self.completed:

        dc.SetTextForeground(wx.Colour('#063d8a'))
        dc.DrawText("Completed", 25, 20)

def OnPaint(self, e):
    """Reacts to paint events"""

    dc = wx.PaintDC(self)
    self.BuildWorld(dc)

def OnKeyDown(self, e):
    """Handles key events"""

    if self.completed:
        return

    key = e.GetKeyCode()

    if key == wx.WXK_LEFT:

        if self.CheckWallCollision(self.soko,
                                     s.LEFT_COLLISION):
            return

        if self.CheckBagCollision(s.LEFT_COLLISION):
            return

```

```

        self.soko.Move(-s.SPACE, 0)

    elif key == wx.WXK_RIGHT:

        if self.CheckWallCollision(self.soko,
                                    s.RIGHT_COLLISION):
            return

        if self.CheckBagCollision(s.RIGHT_COLLISION):
            return

        self.soko.Move(s.SPACE, 0)

    elif key == wx.WXK_UP:

        if self.CheckWallCollision(self.soko,
                                    s.TOP_COLLISION):
            return

        if self.CheckBagCollision(s.TOP_COLLISION):
            return

        self.soko.Move(0, -s.SPACE)

    elif key == wx.WXK_DOWN:

        if self.CheckWallCollision(self.soko,
                                    s.BOTTOM_COLLISION):
            return

        if self.CheckBagCollision(s.BOTTOM_COLLISION):
            return

        self.soko.Move(0, s.SPACE)

    elif key == ord('r') or key == ord('R'):
        self.RestartLevel()

    self.Refresh()

def CheckWallCollision(self, actor, _type):
    """Checks all possible bag collisions"""

    if _type == s.LEFT_COLLISION:

        for i in range(len(self.walls)):
            wall = self.walls[i]
            if actor.IsLeftCollision(wall):
                return True
        return False

    elif _type == s.RIGHT_COLLISION:

        for i in range(len(self.walls)):
            wall = self.walls[i]
            if actor.IsRightCollision(wall):
                return True
        return False

    elif _type == s.TOP_COLLISION:

```

```

        for i in range(len(self.walls)):
            wall = self.walls[i]
            if actor.IsTopCollision(wall):
                return True
        return False

    elif _type == s.BOTTOM_COLLISION:

        for i in range(len(self.walls)):
            wall = self.walls[i]
            if actor.IsBottomCollision(wall):
                return True
        return False

def CheckBagCollision(self, _type):
    """Checks if sokoban collides with a bag"""

    if _type == s.LEFT_COLLISION:

        for i in range(len(self.baggs)):
            bag = self.baggs[i]
            if self.soko.IsLeftCollision(bag):
                for i in range(len(self.baggs)):
                    item = self.baggs[i]
                    if not bag is item:
                        if bag.IsLeftCollision(item):
                            return True
                if self.CheckWallCollision(bag,
                    s.LEFT_COLLISION):
                    return True
            bag.Move(-s.SPACE, 0)
            self.IsCompleted()

        return False

    elif _type == s.RIGHT_COLLISION:

        for i in range(len(self.baggs)):
            bag = self.baggs[i]
            if self.soko.IsRightCollision(bag):
                for i in range(len(self.baggs)):
                    item = self.baggs[i]
                    if not bag is item:
                        if bag.IsRightCollision(item):
                            return True
                if self.CheckWallCollision(bag,
                    s.RIGHT_COLLISION):
                    return True
            bag.Move(s.SPACE, 0)
            self.IsCompleted()

        return False

    elif _type == s.TOP_COLLISION:

        for i in range(len(self.baggs)):
            bag = self.baggs[i]
            if self.soko.IsTopCollision(bag):
                for i in range(len(self.baggs)):
                    item = self.baggs[i]
                    if not bag is item:

```

```

        if bag.IsTopCollision(item):
            return True
        if self.CheckWallCollision(bag,
            s.TOP_COLLISION):
            return True
        bag.Move(0, -s.SPACE)
        self.IsCompleted()

    return False

elif _type == s.BOTTOM_COLLISION:

    for i in range(len(self.baggs)):
        bag = self.baggs[i]
        if self.soko.IsBottomCollision(bag):
            for i in range(len(self.baggs)):
                item = self.baggs[i]
                if not bag is item:
                    if bag.IsBottomCollision(item):
                        return True
            if self.CheckWallCollision(bag,
                s.BOTTOM_COLLISION):
                return True
            bag.Move(0, s.SPACE)
            self.IsCompleted()

    return False

def IsCompleted(self):
    """Checks whether a task was completed"""

    num = len(self.baggs)
    completed = 0

    for i in range(num):

        bag = self.baggs[i]
        for j in range(num):
            area = self.areas[j]
            if bag.x() == area.x() and \
                bag.y() == area.y():
                completed += 1

    if completed == num:
        self.completed = True
        self.Refresh()

def RestartLevel(self):
    """Restarts level"""

    del self.areas[:]
    del self.baggs[:]
    del self.walls[:]

    self.InitWorld()

    if self.completed:
        self.completed = False

```

```

class Sokoban(wx.Frame):

    def __init__(self, *args, **kw):
        super(Sokoban, self).__init__(*args, **kw)

        self.InitUI()
        self.SetSize((450, 300))
        self.SetTitle("Sokoban")
        self.Centre()

    def InitUI(self):

        self.board = Board(self)

def main():

    app = wx.App()
    sokoban = Sokoban(None)
    sokoban.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

The game is simplified; it only provides the very basic functionality. The game has one level.

```

class s:
    """Stores constants to avoid global values"""

    SPACE = 20
    OFFSET = 30

    LEFT_COLLISION = 1
    RIGHT_COLLISION = 2
    TOP_COLLISION = 3
    BOTTOM_COLLISION = 4

```

Inside the s class, we store the game constants. The wall image size is 20x20 px; therefore the `SPACE` constant is set to 20. The `OFFSET` is the initial distance between the borders of the window and the game world. There are four types of collisions. Each one is represented by a numerical constant.

```

    level = """
    #####+
    ##  #+
    ## $  #+
    ##### $##+
    ##  $ $  #+
    #### # ## # #####+
    ##  # ## #####  . .#+
    ## $ $  . .#+
    ##### ## # @##  . .#+
    ##      #####+
    ######+
    """

```

This is the level of the game. Except for the space, there are five characters. The hash (#) stands for a wall. The dollar (\$) represents the box to move. The dot (.) character represents the place where we must move the box. The at (@) character is the sokoban. Finally, the (+) marks the end of a line.

```
class Actor:
    """Generic actor class for all game objects"""

    def __init__(self, x, y):
        self._x = x
        self._y = y
    ...
```

This is the constructor of the `Actor` class. The class is a base class for other classes in the game. It encapsulates the basic functionality of an object in the Sokoban game.

```
def GetImage(self):
    return self.image

def x(self):
    return self._x

def y(self):
    return self._y
```

These are generic methods that return the image and the coordinates of an object.

```
def IsLeftCollision(self, actor):
    if ((self.x() - s.SPACE) == actor.x()) and \
        (self.y() == actor.y()):
        return True
    else:
        return False
```

The `IsLeftCollision` method checks if the object collides with another object on the left side.

```
class Player(Actor):

    def __init__(self, x, y):
        super(Player, self).__init__(x, y)

        try:
            self.image = wx.Bitmap("sokoban.png")

        except Exception as e:

            print(e.message)
            sys.exit(1)

    def Move(self, x, y):
        self._x += x
        self._y += y
```

The `Player` class inherits functionality from the base `Actor` class. It loads an image which represents the sokoban object in the game. It has an additional `Move` method. The method is responsible for moving an object.

```
def InitVariables(self):
    '''Initiates game variables'''

    self.world = []
    self.walls = []
    self.baggs = []
    self.areas = []

    self.w = 0
    self.h = 0

    self.completed = False
    self.soko = None
```

In the `InitVariables` method, we initiate the game variables. The `world` list stores all objects of the game. The `walls`, `baggs`, `areas` store all the walls, bags, and areas of the game. An area is a place where we must place the baggage. The `w` and `h` variables store the width and height of the game world. The `completed` variable is used to determine whether we have already finished the. Finally, the `soko` variable stores the sokoban player.

```
def InitWorld(self):
    '''Sets up the game world'''

    x = s.OFFSET
    y = s.OFFSET
    ...
```

The `InitWorld` method initiates the game world. It goes through the level string and fills the above mentioned lists.

```
elif item == '$':
    b = Baggage(x, y)
    self.baggs.append(b)
    x += s.SPACE
```

In case of the dollar character, we create a `Baggage` object. The object is appended to the `baggs` list. The `x` variable is increased accordingly.

```
def BuildWorld(self, dc):
    """Constructs the game world"""
    ...
```

The `BuildWorld` method draws the game world on the window.

```
world = self.walls + self.areas + \
        self.baggs + [self.soko]
```

We create a world list which includes all objects of the game.

```
for i in range(len(world)):

    item = world[i]

    if isinstance(item, Player) or \
        isinstance(item, Baggage):
        dc.DrawBitmap(item.GetImage(), item.x()+2,
                       item.y()+2)
```



```

else:
    dc.DrawBitmap(item.GetImage(), item.x(),
        item.y())

```

We iterate through the world list and draw the objects. The player and the baggage images are a bit smaller. We add 2 px to their coordinates to center them.

```

if self.completed:
    dc.DrawText("Completed", 25, 20)

```

If the level is completed, we draw "Completed" in the upper left corner of the window.

```

if key == wx.WXK_LEFT:

    if self.CheckWallCollision(self.soko,
        s.LEFT_COLLISION):
        return

    if self.CheckBagCollision(s.LEFT_COLLISION):
        return

    self.soko.Move(-s.SPACE, 0)

```

Inside the `OnKeyDown` method, we check what keys were pressed. We can control the sokoban object with the cursor keys. If we press the left cursor key, we check if the sokoban collides with a wall or with a baggage. If it does not, we move the sokoban to the left.

```

elif key == ord('r') or key == ord('R'):
    self.RestartLevel()

```

Sometimes we move a baggage to a position that the game cannot be finished. Therefore, we have an option to restart the level. The level is restarted with the R key.

```

def CheckWallCollision(self, actor, _type):
    '''Checks for a collision of sokoban with a wall'''

    if _type == s.LEFT_COLLISION:

        for i in range(len(self.walls)):
            wall = self.walls[i]
            if actor.IsLeftCollision(wall):
                return True
        return False
    ...

```

The `CheckWallCollision` method was created to ensure that the sokoban or a baggage do not pass the wall. There are four types of collisions. The above lines check for a left collision.

```

def CheckBagCollision(self, _type):
    '''Checks if sokoban collides with a bag'''

    if _type == s.LEFT_COLLISION:

```

```

        for i in range(len(self.baggs)):
            bag = self.baggs[i]
            if self.soko.IsLeftCollision(bag):
                for i in range(len(self.baggs)):
                    item = self.baggs[i]
                    if not bag is item:
                        if bag.IsLeftCollision(item):
                            return True
                if self.CheckWallCollision(bag,
                    s.LEFT_COLLISION):
                    return True
            bag.Move(-s.SPACE, 0)
            self.IsCompleted()

    return False
...

```

The `CheckBagCollision` is a bit more involved. A baggage can collide with a wall, with a sokoban object, or with another baggage. The baggage can be moved only if it collides with a sokoban and does not collide with another baggage or a wall. When the baggage is moved, it is time to check if the level is completed by calling the `IsCompleted` method.

```

def IsCompleted(self):
    """Checks whether a task was completed"""

    num = len(self.baggs)
    completed = 0

    for i in range(num):

        bag = self.baggs[i]
        for j in range(num):
            area = self.areas[j]
            if bag.x() == area.x() and \
                bag.y() == area.y():
                completed += 1

    if completed == num:
        self.completed = True
        self.Refresh()

```

The `IsCompleted` method checks if the level is completed. We get the number of bags. We compare the x, y coordinates of all the bags and the destination areas. The game is finished when the completed variable equals the number of bags in the game.

```

def RestartLevel(self):
    """Restarts level"""

    del self.areas[:]
    del self.baggs[:]
    del self.walls[:]

    self.InitWorld()

    if self.completed:
        self.completed = False

```

If we do some bad move, we can restart the level. We delete all objects from

the important lists and initiate the world again. The completed variable is set to False.

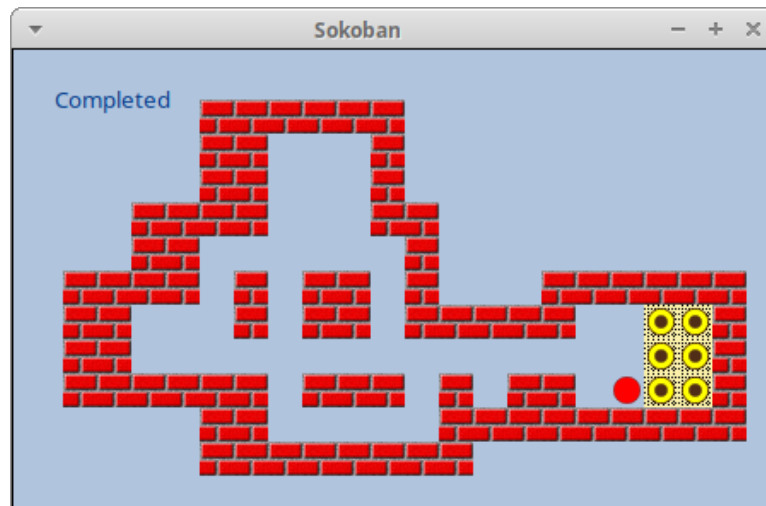


Figure 10.1: Sokoban game

Figure 10.1 shows a screenshot of a finished Sokoban game.

Chapter 11

Minesweeper

Minesweeper is a popular board game shipped with many operating systems. The goal of the game is to sweep all mines from a minefield. If a player clicks on the cell which contains a mine, the mine detonates and the game is over. Further, a cell can contain a number or it can be blank. The number indicates how many mines are adjacent to this particular cell. We set a mark on a cell by right clicking on it. This way we indicate we believe, there is a mine.

Listing 11.1: Minesweeper game

```
#!/usr/bin/env python

"""
ZetCode Advanced wxPython tutorial

This is a simple Minesweeper game
clone.

Author: Jan Bodnar
Website: zetcode.com
"""

import random
import sys
import wx

class m:
    """Stores constants and variables to avoid
    global values"""

    WIDTH = 250
    HEIGHT = 290

    CELL_SIZE = 15
    NUM_IMAGES = 13

    COVER_FOR_CELL = 10
    MARK_FOR_CELL = 10
    EMPTY_CELL = 0
    MINE_CELL = 9

    COV_MINE_CELL = MINE_CELL + COVER_FOR_CELL
```

```

MARKED_MINE_CELL = COV_MINE_CELL + MARK_FOR_CELL

DRAW_MINE = 9
DRAW_COVER = 10
DRAW_MARK = 11
DRAW_WRONG_MARK = 12

img = [0] * NUM_IMAGES

class Board(wx.Panel):

    def __init__(self, parent, statusbar):
        super(Board, self).__init__(parent)

        self.statusbar = statusbar

        self.SetFocus()
        self.InitGame()
        self.NewGame()

    def InitGame(self):
        """Initiates the game"""

        self.SetDoubleBuffered(True)

        self.fld = []
        self.inGame = False
        self.mns_left = 0

        self.mines = 40
        self.rows = 16
        self.cols = 16
        self.all_cells = 0

        try:
            for i in range(m.NUM_IMAGES):
                m.img[i] = wx.Bitmap(str(i) + ".png")

        except Exception as e:

            print(e.message)
            sys.exit(1)

        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_LEFT_DOWN, self.OnMouseDown)
        self.Bind(wx.EVT_RIGHT_DOWN, self.OnMouseDown)

    def NewGame(self):
        """Starts a new game"""

        i = 0
        position = 0
        cell = 0

        self.inGame = True
        self.mns_left = self.mines

        self.all_cells = self.rows * self.cols
        self.fld = [0] * self.all_cells

```

```

for i in range(self.all_cells):
    self.fld[i] = m.COVER_FOR_CELL

self.statusbar.SetStatusText(str(self.mns_left))

i = 0
while i < self.mines:

    position = random.randint(0, self.all_cells)

    if position < self.all_cells and \
        self.fld[position] != m.COV_MINE_CELL:

        current_col = position % self.cols
        self.fld[position] = m.COV_MINE_CELL
        i = i + 1

    if current_col > 0:
        cell = position - 1 - self.cols
        if cell >= 0:
            if self.fld[cell] != m.COV_MINE_CELL:
                self.fld[cell] += 1
        cell = position - 1
        if cell >= 0:
            if self.fld[cell] != m.COV_MINE_CELL:
                self.fld[cell] += 1

        cell = position + self.cols - 1
        if cell < self.all_cells:
            if self.fld[cell] != m.COV_MINE_CELL:
                self.fld[cell] += 1

    cell = position - self.cols
    if cell >= 0:
        if self.fld[cell] != m.COV_MINE_CELL:
            self.fld[cell] += 1
    cell = position + self.cols
    if cell < self.all_cells:
        if self.fld[cell] != m.COV_MINE_CELL:
            self.fld[cell] += 1

    if current_col < self.cols - 1:
        cell = position - self.cols + 1
        if cell >= 0:
            if self.fld[cell] != m.COV_MINE_CELL:
                self.fld[cell] += 1
        cell = position + self.cols + 1
        if cell < self.all_cells:
            if self.fld[cell] != m.COV_MINE_CELL:
                self.fld[cell] += 1
        cell = position + 1
        if cell < self.all_cells:
            if self.fld[cell] != m.COV_MINE_CELL:
                self.fld[cell] += 1

def FindEmptyCell(self, j):
    """Recursively finds all empty cells"""

```

```

current_col = j % self.cols
cell = 0

if current_col > 0:
    cell = j - self.cols - 1
    if cell >= 0:
        if self.fld[cell] > m.MINE_CELL:
            self.fld[cell] -= m.COVER_FOR_CELL
        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

    cell = j - 1
    if cell >= 0:
        if self.fld[cell] > m.MINE_CELL:
            self.fld[cell] -= m.COVER_FOR_CELL
        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

    cell = j + self.cols - 1
    if cell < self.all_cells:
        if self.fld[cell] > m.MINE_CELL:
            self.fld[cell] -= m.COVER_FOR_CELL
        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

cell = j - self.cols
if cell >= 0:
    if self.fld[cell] > m.MINE_CELL:
        self.fld[cell] -= m.COVER_FOR_CELL
    if self.fld[cell] == m.EMPTY_CELL:
        self.FindEmptyCell(cell)

cell = j + self.cols
if cell < self.all_cells:
    if self.fld[cell] > m.MINE_CELL:
        self.fld[cell] -= m.COVER_FOR_CELL;
    if self.fld[cell] == m.EMPTY_CELL:
        self.FindEmptyCell(cell)

if current_col < self.cols - 1:
    cell = j - self.cols + 1
    if cell >= 0:
        if self.fld[cell] > m.MINE_CELL:
            self.fld[cell] -= m.COVER_FOR_CELL
        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

    cell = j + self.cols + 1
    if cell < self.all_cells:
        if self.fld[cell] > m.MINE_CELL:
            self.fld[cell] -= m.COVER_FOR_CELL
        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

cell = j + 1
if cell < self.all_cells:
    if self.fld[cell] > m.MINE_CELL:

```

```

        self.fld[cell] -= m.COVER_FOR_CELL
        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

def OnPaint(self, e):
    """Reacts to paint event"""

    dc = wx.PaintDC(self)
    self.DrawBoard(dc)

def DrawBoard(self, dc):
    """Draws images on the board"""

    cell = 0
    uncover = 0

    for i in range(self.rows):
        for j in range(self.cols):

            cell = self.fld[(i * self.cols) + j]

            if not self.inGame:
                if cell == m.COV_MINE_CELL:
                    cell = m.DRAW_MINE
                elif cell == m.MARKED_MINE_CELL:
                    cell = m.DRAW_MARK
                elif cell > m.COV_MINE_CELL:
                    cell = m.DRAW_WRONG_MARK
                elif cell > m.MINE_CELL:
                    cell = m.DRAW_COVER

            else:
                if cell > m.COV_MINE_CELL:
                    cell = m.DRAW_MARK
                elif cell > m.MINE_CELL:
                    cell = m.DRAW_COVER

            dc.DrawBitmap(m.img[cell], j*m.CELL_SIZE,
                          i*m.CELL_SIZE)

def OnMouseDown(self, e):
    """Reacts to mouse click events"""

    x = e.GetX()
    y = e.GetY()

    cCol = x // m.CELL_SIZE
    cRow = y // m.CELL_SIZE

    rep = False

    if not self.inGame:
        self.NewGame()
        self.Refresh()
        return

    pos = (cRow * self.cols) + cCol

```



```

        if x < self.cols * m.CELL_SIZE and \
            y < self.rows * m.CELL_SIZE:

            if e.RightDown():

                if self.fld[pos] > m.MINE_CELL:
                    rep = True

                if self.fld[pos] <= m.COV_MINE_CELL:
                    if self.mns_left > 0:
                        self.fld[pos] += m.MARK_FOR_CELL
                        self.mns_left = self.mns_left - 1
                        msg = str(self.mns_left)
                        self.statusbar.SetStatusText(msg)
                    else:
                        msg = "No marks left"
                        self.statusbar.SetStatusText(msg)
                else:
                    self.fld[pos] -= m.MARK_FOR_CELL
                    self.mns_left = self.mns_left + 1
                    msg = str(self.mns_left)
                    self.statusbar.SetStatusText(msg)

            else:

                if self.fld[pos] == m.MARKED_MINE_CELL:
                    return

                if self.fld[pos] > m.MINE_CELL and \
                    self.fld[pos] < m.MARKED_MINE_CELL:

                    self.fld[pos] -= m.COVER_FOR_CELL
                    rep = True

                if self.fld[pos] == m.MINE_CELL:

                    self.inGame = False
                    msg = "Game lost"
                    self.statusbar.SetStatusText(msg)

                if self.fld[pos] == m.EMPTY_CELL:
                    self.FindEmptyCell(pos)

            uncovered = 0
            for cell in range(self.all_cells):
                if self.fld[cell] < m.MINE_CELL:
                    uncovered = uncovered + 1

            if self.mns_left == 0 and uncovered == \
                (self.all_cells - self.mines):
                self.inGame = False
                self.statusbar.SetStatusText("Game won")

        if rep:
            self.Refresh()

class Mines(wx.Frame):

    def __init__(self, *args, **kw):
        super(Mines, self).__init__(*args, **kw)

```

```

        self.InitUI()
        self.SetSize((m.WIDTH, m.HEIGHT))
        self.SetTitle("Minesweeper")
        self.Centre()

    def InitUI(self):

        self.statusbar = self.CreateStatusBar()
        self.board = Board(self, self.statusbar)
        self.SetMinSize((m.WIDTH, m.HEIGHT))
        self.SetMaxSize((m.WIDTH, m.HEIGHT))

def main():

    app = wx.App()
    mines = Mines(None)
    mines.Show()
    app.MainLoop()

if __name__ == "__main__":
    main()

```

This is the code for the Minesweeper game in wxPython.

```

CELL_SIZE = 15
NUM_IMAGES = 13

```

There are 13 images used in this game. A cell can be surrounded by maximum of 8 mines, so we need image numbers 1..8. We need images for an empty cell, a mine, a covered cell, a marked cell and finally for a wrongly marked cell. The size of each of the images is 15x15px.

```

COVER_FOR_CELL = 10
MARK_FOR_CELL = 10
EMPTY_CELL = 0
...

```

A mine field is a list of numbers. For example, 0 denotes an empty cell. Number 10 is used for a cell cover as well as for a mark. Using constants improves readability of the code.

```

DRAW_MINE = 9
DRAW_COVER = 10
DRAW_MARK = 11
DRAW_WRONG_MARK = 12

```

These constants are used inside the `DrawBoard` method to draw the cells of the minefield. They are indexes into the `img` image list.

```

self.fld = []

```

The `fld` is a list of numbers. Each cell in the field has a specific number; e.g.

a mine cell has number 9. A cell that is adjacent to two mines has number 2. The numbers are added. For example, a covered mine has number 19: 9 for the mine and 10 for the cell cover.

```
self.mines = 40
self.rows = 16
self.cols = 16
```

The minefield in our game has 40 hidden mines. There are 16 rows and 16 columns in this field. So there are 256 cells together in the minefield.

```
for i in range(m.NUM_IMAGES):
    m.img[i] = wx.Bitmap(str(i) + ".png")
```

Here we load our images into the `m.img` list. The images are named 0.png, 1.png ... 12.png.

```
self.all_cells = self.rows * self.cols
self.fld = [0] * self.all_cells
```

```
for i in range(self.all_cells):
    self.fld[i] = m.COVER_FOR_CELL
```

In the `NewGame` method, we start the game. We set up the minefield. The for loop creates a cover for each cell.

```
i = 0
while i < self.mines:

    position = random.randint(0, self.all_cells)

    if position < self.all_cells and \
        self.fld[position] != m.COV_MINE_CELL:

        current_col = position % self.cols
        self.fld[position] = m.COV_MINE_CELL
        i = i + 1
```

In the while cycle, we randomly position all mines on the field. In other words, we randomly add forty `m.COV_MINE_CELL` numbers to the field list.

```
if current_col > 0:
    cell = position - 1 - self.cols
    if cell >= 0:
        if self.fld[cell] != m.COV_MINE_CELL:
            self.fld[cell] += 1
```

After we have added a new mine to the field, we must create numbers around the mine. Each of the cells is surrounded by 8 cells. (This does not apply to the border cells.) We raise the number for adjacent cells for the randomly placed mine. A mine can be placed next to another mine; therefore we check that the adjacent cell is not a covered mine. In the code excerpt, we add 1 to the top neighbour of the cell in question.

```
cell = j - 1
if cell >= 0:
    if self.fld[cell] > m.MINE_CELL:
        self.fld[cell] -= m.COVER_FOR_CELL
```

```

        if self.fld[cell] == m.EMPTY_CELL:
            self.FindEmptyCell(cell)

```

In the `FindEmptyCell` method, we find empty cells. If a player clicks on a mine cell, the game is over. If he clicks on a cell adjacent to a mine, he uncovers a number indicating how many mines the cell is adjacent to. Clicking on an empty cell leads to uncovering many other empty cells plus cells with a number that form a border around empty cells. We use a recursive algorithm to find empty cells. In the above code, we check the cell that is located to the left of an empty cell in question. If it is not empty, we uncover it. If it is empty, we repeat the whole process by recursively calling the `FindEmptyCell` method.

```

...
else:
    if cell > m.COV_MINE_CELL:
        cell = m.DRAW_MARK
    elif cell > m.MINE_CELL:
        cell = m.DRAW_COVER

dc.DrawBitmap(m.img[cell], j*m.CELL_SIZE,
              i*m.CELL_SIZE)

```

The `DrawBoard` method turns numbers into images. In the above code, we check what number is in the current cell of the field. Depending on this number, we draw an appropriate image for the cell.

```

def OnMouseDown(self, e):
    """Reacts to mouse click events"""
    ...

```

In the `OnMouseDown` method, we react to mouse clicks. The Minesweeper game is controlled solely by mouse. We react to left and right mouse clicks.

```

x = e.GetX()
y = e.GetY()

cCol = x // m.CELL_SIZE
cRow = y // m.CELL_SIZE

```

First, we retrieve the x and y coordinates of a mouse click. If we divide these two numbers by the cell size, we get the current column and current row of the field.

```

rep = False

```

The `rep` variable controls the repainting of the board. The Minesweeper game is mostly static. For example, we do not need to repaint the board if we have clicked on an uncovered cell or left clicked on a marked cell.

```

pos = (cRow * self.cols) + cCol

```

Using the current row and the current column, we calculate a position into the field. This position is an index used to get the cell on which we have clicked.

```

if x < self.cols * m.CELL_SIZE and \
   y < self.rows * m.CELL_SIZE:

```

We continue only if we have clicked inside the minefield.

```
if self.fld[pos] <= m.COV_MINE_CELL:
    if self.mns_left > 0:
        self.fld[pos] += m.MARK_FOR_CELL
        self.mns_left = self.mns_left-1
        msg = str(self.mns_left)
        self.statusbar.SetStatusText(msg)
    else:
        msg = "No marks left"
        self.statusbar.SetStatusText(msg)
```

These lines determine what happens if we right click on a cell that is not already marked by a user to be a mine cell. If there are still some mines left, we add a `m.MARK_FOR_CELL` value to a current cell, decrease the `mns_left` and update the statusbar. Otherwise, we show a "No marks left" message in the statusbar.

```
else:
    self.fld[pos] -= m.MARK_FOR_CELL
    self.mns_left = self.mns_left+1
    msg = str(self.mns_left)
    self.statusbar.SetStatusText(msg)
```

We can change our mind as for whether a cell is supposed to be a mine cell. By right clicking on a marked cell, we remove the mark and increase the `mns_left` variable.

```
if self.fld[pos] == m.MARKED_MINE_CELL:
    return
```

Nothing happens if we left click on the marked cell. If we want to uncover a marked cell, we first have to remove the mark with another right click.

```
self.fld[pos] -= m.COVER_FOR_CELL
```

A left click removes a cover from the cell.

```
if self.fld[pos] == m.MINE_CELL:

    self.inGame = False
    msg = "Game lost"
    self.statusbar.SetStatusText(msg)
```

If we uncover a mine, the game is over.

```
if self.fld[pos] == m.EMPTY_CELL:
    self.FindEmptyCell(pos)
```

If we click on an empty cell, we call the `FindEmptyCell` method which recursively looks for all adjacent empty cells.

```
uncovered = 0
for cell in range(self.all_cells):
    if self.fld[cell] < m.MINE_CELL:
        uncovered = uncovered + 1

    if self.mns_left == 0 and uncovered == \
        (self.all_cells - self.mines):
```

```

self.inGame = False
self.statusbar.SetStatusText("Game won")

```

These lines check for successful game over. Cells with numbers lower than `m.MINE_CELL` are either empty cells or cells which are adjacent to a mine, having numbers 1..8. All these cells, without a cover, form a list of uncovered cells. The game is won if there are no more (marked) mines left and the number of uncovered cells is equal to all cells minus mine cells.



Figure 11.1: Minesweeper game

Figure 11.1 shows a screenshot of a finished Minesweeper game.

Index

- basename, 94
- Blit, 167
- Custom widgets, 50–74
 - ColourWheel widget, 62
 - ProgressMeter widget, 50
 - Thermometer widget, 54
- Games
 - Minesweeper, 219–229
 - Snake, 195–204
 - Sokoban, 205–218
- GetClientSize, 202
- GetTextExtent, 202
- Images, 35–49
 - art provider, 40
 - Blur, 36
 - blurring, 35
 - Copy, 44
 - cropping, 38
 - embedding, 39
 - GetBitmap, 40
 - GetSubBitmap, 39
 - grayscale, 42
 - ScaleImage, 37
 - scaling, 36
 - screenshot, 46
 - watermark, 45
 - wx.ART_FILE_OPEN, 41
 - wx.ArtProvider.GetBitmap, 41
 - wx.Bitmap, 35, 37, 43
 - wx.BitmapFromImage, 36
 - wx.DisplaySize, 49
 - wx.Image, 35–37, 42
 - wx.ImageFromBitmap, 36
 - wx.StaticBitmap, 35–37, 40
- Layout management, 7–34
 - absolute positioning, 7, 8
 - wx.BoxSizer, 9–22
 - wx.GridBagSizer, 22–34
 - maxsize, 98
 - Skip, 152
- wx.AcceleratorTable, 101
- wx.Bitmap, 36, 37, 45, 167, 186, 188, 201
- wx.BoxSizer
 - alignment, 15
 - buttons example, 16
 - gaps, 12
 - nesting, 10
 - new folder example, 18
 - proportion, 13
 - row of buttons, 10
 - windows example, 20
- wx.EVT_KEY_DOWN, 200
- wx.EVT_PAINT, 200
- wx.EVT_TIMER, 200
- wx.EVT_TREE_ITEM_EXPANDING, 143
- wx.FD_OVERWRITE_PROMPT, 94
- wx.FD_SAVE, 94
- wx.FileDialog, 93
 - GetPath, 93
- wx.grid.Grid, 145–175
 - basics, 145
 - cell attribute, 147
 - cell editor, 163
 - cell renderers, 164
 - CreateGrid, 146, 150
 - cursor, 151
 - custom cell renderer, 165
 - EnableGridLines, 151
 - Enter key, 153
 - GetCellSize, 162
 - GetCol, 152
 - GetGridCursorCol, 158, 163
 - GetGridCursorRow, 158, 163
 - GetNumberCols, 158
 - GetNumberRows, 159
 - GetRow, 152

- GetSelectedCells, 155, 158
- GetSelectedCols, 155, 159
- GetSelectedRows, 155, 158
- GetSelectionBlockBottomRight, 155
- GetSelectionBlockTopLeft, 155
- grid lines, 148
- GridCellAttr, 147
- GridCellBoolEditor, 164
- GridCellBoolRenderer, 165
- GridCellDateRenderer, 165
- GridCellFloatRenderer, 165
- GridCellNumberEditor, 164
- GridCellNumberRenderer, 165
- GridCellRenderer, 165, 167
- GridCellStringRenderer, 165
- GridCellTextEditor, 164
- GridTableBase, 145, 170
- HideColLabels, 150
- HideRowLabels, 151
- merge cells, 159
- model & view, 168, 171
- MoveCursorDown, 155
- MoveCursorRight, 155
- row and column labels, 148
- selection, 155
- SetCellBackgroundColour, 146
- SetCellEditor, 163
- SetCellFont, 146
- SetCellRenderer, 165, 168
- SetCellSize, 146, 159, 162
- SetCellTextColour, 146
- SetCellValue, 146
- SetColLabelSize, 150
- SetColSize, 168
- SetGridCursor, 152
- SetRowLabelSize, 151
- SetRowSize, 168
- SetTable, 171
- wx.GridEvent, 152
- wx.GridBagSizer
 - alignment, 26
 - gaps, 23
 - growable rows & columns, 28
 - new folder example, 30
 - simple example, 22
 - span, 24
 - Windows example, 32
- wx.ImageList
 - Add, 140
- wx.ListCtrl, 96–121
 - AssignImageList, 110
 - auto width mixin, 104
 - background color, 104
 - CheckItem, 113
 - ColumnSorterMixin, 114, 116, 117
 - editable, 102
 - EnableCheckBoxes, 111
 - GetIndex, 98
 - GetItem, 99
 - GetItemCount, 113
 - GetListCtrl, 114
 - GetNextSelected, 101
 - GetSelectedItemCount, 102
 - icon view, 108
 - images, 106
 - InsertColumn, 98
 - InsertImageItem, 110
 - InsertItem, 98
 - itemDataMap, 114, 117
 - ListCtrlAutoWidthMixin, 105
 - OnGetItemAttr, 118
 - OnGetItemColumnImage, 118
 - OnGetItemImage, 118
 - report view, 96
 - selection, 99
 - SetItem, 98
 - SetItemBackgroundColour, 105
 - SetItemCount, 118, 120
 - SetItemData, 114, 117
 - SetItemImage, 108
 - virtual, 118
 - wx.LC_ICON, 108, 110
 - wx.LC_VIRTUAL, 118, 120
- wx.Menu, 78, 92, 161
 - AppendCheckItem, 150
 - AppendRadioItem, 155
 - wx.EVT_MENU_OPEN, 162
- wx.MenuBar, 78, 150, 161
- wx.NewIdRef, 78, 101, 126, 150, 154, 161
- wx.richtext.RichTextCtrl, 176–193
 - AppendText, 176
 - ApplyBoldToSelection, 186
 - BeginBold, 178
 - BeginItalic, 178
 - BeginNumberedBullet, 181
 - BeginTextColour, 179
 - BeginURL, 183
 - bullets, 179
 - EndBold, 178

- EndFontSize, 179
- EndItalic, 178
- EndTextColour, 179
- EndURL, 183
- font weight & font style, 184
- Freeze, 180
- GetNumberOfLines, 177
- GetValue, 177
- images & URLs, 181
- introductory example, 176
- IsSelectionBold, 186
- load file from XML, 191
- LoadFile, 193
- NewLine, 178
- ProcessEvent, 188
- RichTextXMLHandler, 189
- save text in XML, 189
- SetFilename, 190
- text methods, 177
- Thaw, 180
- undo & redo, 187
- WriteImageFile, 183
- WriteText, 178
- wx.EVT_TOOL, 188
- wx.EVT_UPDATE_UI, 186, 188
- wx.ID_REDO, 188
- wx.ID_UNDO, 188
- wx.richtext.RICHTEXT_TYPE_XML, 191
- wx.TE_RICH, 84, 86
- wx.TextCtrl, 76–95, 113
 - find, 87
 - GetInsertionPoint, 82
 - GetSelection, 83
 - GetStringSelection, 84
 - lines and columns, 79
 - load/save example, 87
 - PositionToXY, 82
 - selections, 82
 - SetStyle, 86
 - simple example, 76
 - text search example, 84
 - wx.TextAttr, 84
- wx.TreeCtrl, 122–144
 - AddRoot, 123, 143
 - AppendItem, 124, 129
 - AssignImageList, 140
 - GetFirstChild, 129
 - GetItemText, 124, 126, 138
 - GetNextSibling, 129
 - GetRootItem, 129
 - GetSelections, 126
 - images, 139
 - IsOk, 129
 - lazy evaluation, 141
 - OnCompareItems, 134, 138
 - search, 130
 - selection, 124
 - SetItemData, 143
 - SetItemHasChildren, 143
 - SetItemImage, 140
 - simple example, 122
 - sort, 134
 - SortChildren, 134, 138
 - traverse items, 127
 - wx.EVT_TREE_SEL_CHANGED, 123
 - wx.TR_DEFAULT_STYLE, 122
 - wx.TR_HAS_BUTTONS, 126
 - wx.TR_LINES_AT_ROOT, 126
 - wx.TR_MULTIPLE, 126
 - wx.TreeItemId, 126
 - wx.WXK_RETURN, 155