Quick answers to common problems

wxPython 2.8
Application Development Cookbook

Quickly create robust, reliable, and reusable wxPython applications

Cody Precord

[PACKT] PUBLISHING open source community experience distilled

wxPython 2.8
Application Development Cookbook

Quickly create robust, reliable, and reusable wxPython applications

Cody Precord

[PACKT] open source
community experience distilled
PUBLISHING

BIRMINGHAM - MUMBAI

wxPython 2.8
Application Development Cookbook

First published: December 2010

Production Reference: 1031210

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849511-78-0

www.packtpub.com

Cover Image by Vinayak Chittar (vinayak.chittar@gmail.com)

Credits

Author
Cody Precord

Reviewers
Maurice HT Ling
Steve McMahon
Jeff McNeil
Chukwudi Nwachukwu

Acquisition Editor
Steven Wilding

Development Editor
Maitreya Bhakal

Technical Editor
Conrad Sardinha

Indexers
Tejal Daruwale
Rekha Nair

Editorial Team Leader
Akshara Aware

Project Team Leader
Lata Basantani

Project Coordinator
Vincila Colaco

Proofreader
Dirk Manuel

Graphics
Nilesh Mohite

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Author

Cody Precord is a Software Engineer based in Minneapolis, MN, USA. He has been designing and writing systems and application software for AIX, Linux, Windows, and Macintosh OS X for the last ten years using primarily C, C++, Perl, Bash, Korn Shell, and Python. The constant need of working on multiple platforms naturally led Cody to the wxPython toolkit, which he has been using intensely for that last five years. Cody has been primarily using wxPython for his open source project, Editra, which is a cross-platform development tool. He is interested in promoting cross-platform development practices and improving usability in software.

wxPython 2.8
Application Development Cookbook

Quickly create robust, reliable, and reusable wxPython applications

Cody Precord

[PACKT] open source
community experience distilled
PUBLISHING

BIRMINGHAM - MUMBAI

wxPython 2.8
Application Development Cookbook

First published: December 2010

Production Reference: 1031210

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849511-78-0

www.packtpub.com

Cover Image by Vinayak Chittar (vinayak.chittar@gmail.com)

Credits

Author
Cody Precord

Reviewers
Maurice HT Ling
Steve McMahon
Jeff McNeil
Chukwudi Nwachukwu

Acquisition Editor
Steven Wilding

Development Editor
Maitreya Bhakal

Technical Editor
Conrad Sardinha

Indexers
Tejal Daruwale
Rekha Nair

Editorial Team Leader
Akshara Aware

Project Team Leader
Lata Basantani

Project Coordinator
Vincila Colaco

Proofreader
Dirk Manuel

Graphics
Nilesh Mohite

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Author

Cody Precord is a Software Engineer based in Minneapolis, MN, USA. He has been designing and writing systems and application software for AIX, Linux, Windows, and Macintosh OS X for the last ten years using primarily C, C++, Perl, Bash, Korn Shell, and Python. The constant need of working on multiple platforms naturally led Cody to the wxPython toolkit, which he has been using intensely for that last five years. Cody has been primarily using wxPython for his open source project, Editra, which is a cross-platform development tool. He is interested in promoting cross-platform development practices and improving usability in software.

About the Reviewers

Maurice HT Ling completed his Ph.D. in Bioinformatics and B.Sc.(Hons.) in Molecular and Cell Biology from The University of Melbourne where he worked on microarray analysis and text mining for protein-protein interactions. He is currently an Honorary Fellow of The University of Melbourne, Australia. Maurice holds several Chief Editorships including The Python Papers, iConcept Journal of Computational and Mathematical Biology, and Methods and Cases in Computational, Mathematical, and Statistical Biology. In his free time, Maurice likes to train in the gym, read, and enjoy a good cup of coffee. He is also a Senior Fellow of the International Fitness Association, USA.

Steve McMahon is a Python and Plone developer located in Davis, California. His company, Reid-McMahon, LLC specializes in developing Content Management Systems for non-profit organizations. He's been involved in many aspects of the Plone project, including training and core, installer, and add-on development.

Jeff McNeil cut his teeth during the Internet boom, being one of the first employees at one of the larger web-hosting shops. He's done just about everything from server installs to platform development and software architecture. Technical interests include systems management and doing things Pythonically. Jeff recently joined Google.

Chukwudi Nwachukwu, aka Chux, studied Computer Science at Olabisi Onabanjo University, Nigeria. He has, over the years, worked on both Windows and Linux operating systems. Programming is fun. He had to join the programming wagon because programmers are known to solve problems by making computers do things that they visualize in their minds. He programs in over a dozen languages such as Processing, D, Python, and so on. He loves to travel, discover new places, meet interesting people, and learn new human languages too. You can reach him on `chux@users.berlios.de`. He has worked on Java CourseWare, an in-house Java textbook for teaching students.

I acknowledge the following people, who have stood by me through thick and thin, and without whom I wouldn't have gotten to this point in my life. Chinonye, Chigbonkpa, and Chimenka, my siblings. My mom and dad, Mr. and Mrs. Richard Nwachukwu, for their support. Olugbenga Owolabi, you lead me through the land of programming by helping me know what algorithms are all about. Bertrand Ogu, who has always been there for me, thank you. Tola Johnny Odule, a lecturer in Olabisi Onabanjo University, Nigeria, and the elder brother of Dele Odule, the Nollywood actor. Wale Adewoyin and Shirley Otukpa, by God's grace I expect you guys to walk down the aisle soon. Kenneth Oraegbunam of IITA, Nigeria. Olugbenga Siyanbola and Bukola Ibironke of Lintak Enterprises, Lagos, Nigeria. The Adenekans in NNPC, Abuja: Beatrice, Olukayode, Damilola and Tobilola. Adedayo Adenekan in Lagos and other members of the family. Pastor Femi Adeboye of Prodigy Ventures, Ikorodu, Lagos. Dr. Shola Olalude of Shola Medical Centre, Ikorodu, Lagos: thank you for believing in me. Tobi Ojo in Ibadan. Bro. Williams Anthony, you've acted like a father for me, God bless you. Olowooribi Kolawole Taofeek, you are a friend. Yakubu Friday Kelvin, you are lovely. Olaleye Peace, I love you.

---

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

In today's world of desktop applications, there is a great amount of incentive to be able to develop applications that can run in more than one environment. Currently, there are a handful of options available for cross-platform frameworks to develop desktop applications in Python. wxPython is one such cross-platform GUI toolkit for the Python programming language. It allows Python programmers to create programs with a complete, highly-functional graphical user interface, simply and easily. wxPython code style has changed quite a bit over the years, and has become much more Pythonic. The examples that you will find in this book are fully up-to-date and reflect this change in style. This cookbook provides you with the latest recipes to quickly create robust, reliable, and reusable wxPython applications. These recipes will guide you from writing simple, basic wxPython scripts all the way through complex concepts, and also feature various design approaches and techniques in wxPython.

This book starts off by covering a variety of topics, from the most basic requirements of a wxPython application, to some of the more in-depth details of the inner workings of the framework, laying the foundation for any wxPython application. It then explains event handling, basic and advanced user interface controls, interface design and layout, creating dialogs, components, extending functionality, and so on. We conclude by learning how to build and manage applications for distribution.

For each of the recipes, there is an introductory example, then more advanced examples, along with plenty of example code that shows how to develop and manage user-friendly applications. For more experienced developers, most recipes also include an additional discussion of the solution, allowing you to further customize and enhance the component.

What this book covers

Chapter 1, *Getting Started with wxPython,* introduces you to the basics of creating a wxPython application. The topics covered in this chapter will provide you with the information needed to start building your own applications, as well as some insight into the inner workings and structure of the framework.

Chapter 2, *Responding to Events,* shows how to make use of events to drive an application and allow the user to interact with it through the user interface. This chapter starts with an overview of what events are and how they work, and then continues on to cover how to interact with a number of common events.

Chapter 3, *Basic Building Blocks of a User Interface*, discusses a number of the basic widgets that are critical to the creation of nearly all user interfaces. You will be introduced to the usage of widgets such as Buttons, Menus, and ToolBars in this chapter.

Chapter 4, *Advanced Building Blocks of a User Interface*, introduces you to some of the more advanced widgets available in the wxPython control library. These widgets will allow you to create tabbed interfaces and display more complex types of data in your user interface.

Chapter 5, *Providing Information and Alerting Users,* shows multiple techniques for keeping the users of an application informed about what is going on and to provide them with help on interacting with the various controls in the applications interface. This chapter will show you how to use various tooltip controls, message boxes, and splash screens.

Chapter 6, *Retrieving Information from Users*, covers the use of common dialogs to retrieve information from users in order to perform tasks such as opening files, searching text, and even printing. As a part of the recipes for the usage of `FileDialog` and `FindDialogs` you will create a simple Notepad-like application.

Chapter 7, *Window Layout and Design*, is where you will be introduced to a number of concepts and techniques for designing your user interfaces in wxPython. The majority of this chapter will explain the use of Sizers to allow you to quickly implement cross-platform user interfaces.

Chapter 8, *Drawing to the Screen*, gives an introduction to the basics of how a user interface works, by showing you how to use some of the primitive tools to implement your own custom user interface objects. This chapter will show you how to use Device Contexts to perform custom drawing routines by creating a number of custom display controls.

Chapter 9, *Design Approaches and Techniques*, introduces you to a number of common programming patterns, and explain how to apply them to wxPython applications. The information in this chapter will provide you with an understanding of some strong approaches and techniques to software design that will not only serve you in writing wxPython applications but can also be generally applied to other frameworks as well, to expand your programming toolbox.

Chapter 10, *Creating Components and Extending Functionality*, shows you how to extend the functionality of existing user interface components, as well as how to create your own controls. The recipes in this chapter combine much of the information presented in Chapters 2, 7, 8, and 9 together to create new controls and to enhance the capabilities of some of the more basic ones provided by wxPython.

Chapter 11, Using Threads and Timers to Create Responsive Interfaces, dives into the world of concurrent programming. This chapter shows you how to create multi-threaded applications, and covers the special care that is needed when interacting with the user interface from worker threads in order to create stable and responsive interfaces.

Chapter 12, *Building and Managing Applications for Distribution*, concludes the tour of the wxPython framework by introducing you to some useful recipes for bolstering the infrastructure of any application that will be distributed to end users. This includes how to store configuration information, exception handling, internationalization, and how to create and distribute stand-alone binaries of your application.

What you need for this book

All that you will need to get started with wxPython is a good text editor for editing Python source code. There are a number of choices available, but I will provide a shameless plug for my own application, Editra, here since it is included in the wxPython Docs and Demo package, as well as at `http://editra.org`. It is written in wxPython and provides good syntax highlighting and auto-completion support for Python that will help you in learning the wxPython API.

This book is primarily written for Python 2.5/2.6 and wxPython 2.8, although the content of the book also directly applies to later versions of wxPython as well. The suggested software to install is as follows:

1. Latest version of Python 2.6 (`http://www.python.org/download/releases/2.6/`).
2. Latest version of wxPython 2.8 (`http://www.wxpython.org/download.php`).

Who this book is for

This book is written for Python programmers wanting to develop GUI applications. A basic knowledge of Python and object oriented programming concepts is required.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "The `App` object also maintains the `MainLoop`, which is used to drive a wxPython application".

A block of code is set as follows:

```
import wx

class MyApp(wx.App):
    def OnInit(self):
        wx.MessageBox("Hello wxPython", "wxApp")
        return True
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
class MyPanel(wx.Panel):
    __metaclass__ = ClassSynchronizer
    def __init__(self, parent, *args, **kwargs)
```

Any command-line input or output is written as follows:

```
python setup.py py2exe
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Click on **OK** to close it and exit the application".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code for this book

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Getting Started with wxPython

In this chapter, we will cover the components that are at the foundation of nearly all wxPython applications, such as:

- The application object
- The main frame
- Understanding the window hierarchy
- Referencing controls
- Using Bitmaps
- Adding icons to Windows
- Utilizing Stock IDs
- Accessing the clipboard
- Supporting drag and drop
- Two-stage widget creation
- Understanding inheritance limitations

Introduction

In today's world of desktop applications there is a great amount of incentive to be able to develop applications that can run on multiple operating systems and desktop platforms. Currently there are a handful of cross-platform Python frameworks that can be used to develop desktop applications. The wxPython Library is a set of Python bindings to the wxWidgets Library, which is a powerful cross-platform C++ application framework that can be used to create user interfaces. What sets wxPython apart is that, unlike other UI toolkits that draw their own controls, wxPython uses the platform's own native UI toolkit for creating and displaying UI components. This means that a wxPython application will have the same look and feel as other applications on the system since it is using the same controls and themes as the rest of the system.

Developing an application in wxPython provides great flexibility for writing applications that will run on Windows, Macintosh OS X, Linux, and other UNIX like environments. Applications can rapidly be developed on one platform and often deployed to another with little or no changes necessary.

The application object

The `App` object bootstraps the library and initializes the underlying toolkit. All wxPython applications must create an `App` object. This should be instantiated before trying to create any other GUI objects to ensure that all the dependant parts of the library have been properly initialized. The `App` object also maintains the `MainLoop`, which is used to drive a wxPython application.

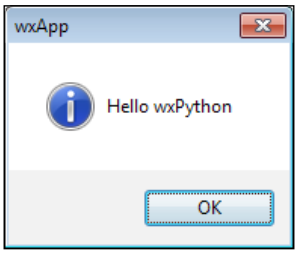
This recipe will demonstrate the basic pattern that all wxPython applications can be built from.

How to do it...

Here we will create a "Hello World" like application to show the basic structure of a wxPython application:

```
import wx

class MyApp(wx.App):
    def OnInit(self):
        wx.MessageBox("Hello wxPython", "wxApp")
        return True

if __name__ == "__main__":
    app = MyApp(False)
    app.MainLoop()
```

Running the previous script will result in the following pop-up dialog shown on the screen. Click on **OK** to close it and exit the application.



How it works...

The application object calls its `OnInit` method when it is created. This method is overridden and used as the main entry point for initializing this application. By returning `True`, the method informs the framework that it is good to go. `OnInit` is where most applications will do their initialization and create their main window(s).

In this example, we created the `App` object by passing `False` as the first argument. This argument is used to tell wxPython whether to redirect output or not. When developing an application, it is advised to always set this to `False`, and to run scripts from the command line so that you can see any error output that might be missed when running the script by double clicking on it.

After creating the application object and once all initializations are complete, the last thing that you need to do is to call the `App` objects `MainLoop` method in order to start the event loop. This method will not return until the last top-level window is destroyed or until the `App` object is told to exit. wxPython is an event-driven system and the `MainLoop` is the heart of the whole system. During each iteration of the loop, events are dispatched to perform all of the tasks in the GUI, such as handling mouse clicks, moving the window, and redrawing the screen.

There's more...

The `wx.App` class constructor has four optional keyword arguments:

```
wx.App(redirect=True, filename=None,
       useBestVisual=False,clearSigInt=True)
```

The four optional keyword arguments are as follows:

- `redirect`: Redirect `stdout`.
- `filename`: If redirect is `True` this can be used to specify an output file to redirect to.
- `useBestVisual`: Specifies whether the application should try to use the best visuals provided by the underlying toolkit. (It does not have an affect on most systems.)
- `clearSigInt`: Should `SIGINT` be cleared? Setting this to `True` will allow the application to be terminated by pressing *Ctrl + C*, like most other applications.

The main frame

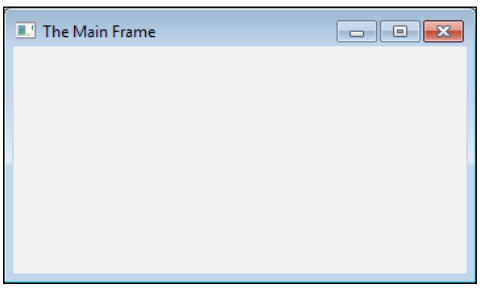
For most applications, you will want to display a window for its users to interact with. In wxPython, the most typical window object is known as a `Frame`. This recipe will show you how to sublass a `Frame` and display it in an application.

How to do it...

This example extends upon the previous recipe to add a minimal empty application window:

```
import wx

class MyApp(wx.App):
    def OnInit(self):
        self.frame = MyFrame(None, title="The Main Frame")
        self.SetTopWindow(self.frame)
        self.frame.Show()

        return True

class MyFrame(wx.Frame):
    def __init__(self, parent, id=wx.ID_ANY, title="",
                 pos=wx.DefaultPosition, size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE,
                 name="MyFrame"):
        super(MyFrame, self).__init__(parent, id, title,
                                      pos, size, style, name)

        # Attributes
        self.panel = wx.Panel(self)

if __name__ == "__main__":
    app = MyApp(False)
    app.MainLoop()
```

Running the previous code will result in a window like the following being shown:



How it works...

The `Frame` is the main top-level window and container for most applications. Let's start by examining our `MyFrame` class. In this class there is one important thing to note. We created a `Panel` object as a child window of the `Frame`. You can think of a `Panel` as a box for containing other controls. Also, in order for a `Frame` to operate and look correct on all platforms, it is important that it has a `Panel` as its main child.

Firstly, in the `OnInit` method of our `App`, we create an instance of `MyFrame`, passing `None` as its first parameter. This parameter is used to specify the parent window of the `Frame`. Because this is our main window, we pass in `None` to indicate that it has no parent. Secondly, we call the `SetTopWindow` method of our `App` in order to set our newly-created `MyFrame` instance as the application's top window. Thirdly and finally, we call `Show` on our `Frame`; this simply does what its name suggests, and shows the `Frame` so that a user can see it, though the `Frame` will not actually be visible on the screen until the `MainLoop` is started.

There's more...

The `Frame` class has a number of style flags that can be set in its constructor to modify the behavior and appearance of the window. These style flags can be combined as a bitmask and are supplied as the value to the constructors' style parameter. The following table outlines some of the common ones. A full list of all available styles can be found in the wxPython online documentation, at `http://wxpython.org/onlinedocs.php`.

Style flags	Description
`wx.DEFAULT_FRAME_STYLE`	This is a bitwise OR of the following flags: ▶ `wx.MINIMIZE_BOX` ▶ `wx.MAXIMIZE_BOX` ▶ `wx.RESIZE_BORDER` ▶ `wx.SYSTEM_MENU` ▶ `wx.CAPTION` ▶ `wx.CLOSE_BOX` ▶ `wx.CLIP_CHILDREN`
`wx.MINIMIZE_BOX`	Display a title bar button that minimizes the Frame
`wx.MAXIMIZE_BOX`	Display a title bar button that maximizes the Frame
`wx.CLOSE_BOX`	Display a title bar button that allows the Frame to be closed. (the "X" button)
`wx.RESIZE_BORDER`	Allow the Frame to be resized by the user when they drag the border
`wx.CAPTION`	Displays a caption on the Frame
`wx.SYSTEM_MENU`	Display a system menu (that is, the menu that is shown when clicking in the frames icon on Windows)
`wx.CLIP_CHILDREN`	Eliminates flicker caused by the background being repainted (Windows only)

Understanding the window hierarchy

All of the different windows and controls in wxPython have a hierarchy of containment. Some controls can be containers for other controls and some cannot. This recipe is geared towards giving an understanding of this hierarchy.

Getting ready

We will be making just a minor change to the `Frame` from the previous recipe, so let's open the code from that recipe to get ready for the new changes.

How to do it...

Here is the new code that will replace our existing Frame class.

```
class MyFrame(wx.Frame):
    def __init__(self, parent, id=wx.ID_ANY, title="",
                 pos=wx.DefaultPosition, size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE,
                 name="MyFrame"):
        super(MyFrame, self).__init__(parent, id, title,
                                      pos, size, style, name)

        # Attributes
        self.panel = wx.Panel(self)
        self.panel.SetBackgroundColour(wx.BLACK)
        self.button = wx.Button(self.panel,
                                label="Push Me",
                                pos=(50, 50))
```

How it works...

Basically, there are three general categories of window objects that are tiered, in the following containment order:

- Top-Level Windows (Frames and Dialogs)
- General Containers (Panels and Notebooks, ...)
- Controls (Buttons, CheckBoxes, ComboBoxes, ...)

The Top-Level Window is at the top of the hierarchy and it can contain any kind of window except another Top-Level Window. General Containers come next, and they can arbitrarily hold any other General Container or Control. Finally, at the bottom of the Hierarchy are the Controls. These are the functional part of a UI that the user will interact with. They can, in some cases, be used to hold other controls, but typically will not. The containment hierarchy is connected to the parental hierarchy of controls. A parent will be the container for its children.

When running the previous sample, this hierarchy becomes apparent. The `Frame`, as we have previously seen, is the outer-most container object; next you can see the `Panel`, which we turned black to make it more visible; finally you can see the `Button`, which was added as a child of the `Panel`.

See also

- The *Referencing controls* recipe in this chapter offers further explanation as to how the window hierarchy is connected together.

Referencing controls

All `Window` objects in an application are connected in various ways. Quite often it is useful to get a reference to an instance of a control so that you can perform some operation on the control or retrieve some data from it. This recipe will show some of the facilities that are available for finding and getting references to controls.

How to do it...

Here we extend the `MyFrame` class from the previous recipe to have an event handler for when its Button is clicked. In the event handler we can see some ways to access different controls in our UI during runtime:

```
class MyFrame(wx.Frame):
    def __init__(self, parent, id=wx.ID_ANY, title="",
                 pos=wx.DefaultPosition, size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE,
                 name="MyFrame"):
        super(MyFrame, self).__init__(parent, id, title,
                                      pos, size, style, name)

        # Attributes
        self.panel = wx.Panel(self)
        self.panel.SetBackgroundColour(wx.BLACK)
        button = wx.Button(self.panel,
                           label="Get Children",
                           pos=(50, 50))
        self.btnId = button.GetId()
```

```
        # Event Handlers
        self.Bind(wx.EVT_BUTTON, self.OnButton, button)

    def OnButton(self, event):
        """Called when the Button is clicked"""
        print "\nFrame GetChildren:"
        for child in self.GetChildren():
            print "%s" % repr(child)

        print "\nPanel FindWindowById:"
        button = self.panel.FindWindowById(self.btnId)
        print "%s" % repr(button)
        # Change the Button's label
        button.SetLabel("Changed Label")

        print "\nButton GetParent:"
        panel = button.GetParent()
        print "%s" % repr(panel)

        print "\nGet the Application Object:"
        app = wx.GetApp()
        print "%s" % repr(app)

        print "\nGet the Frame from the App:"
        frame = app.GetTopWindow()
        print "%s" % repr(frame)
```

How it works...

Each window in the framework keeps a reference to its parent and to its children. Running our program now will print out the results of using the accessor functions that all windows have for finding and retrieving references to their children and other related controls.

- `GetChildren`: This method will return a list of all of the children that the given control has
- `FindWindowById`: This can be used to find a specific child window by using its ID
- `GetParent`: This method will retrieve the window's parent window
- `wx.GetApp`: This is a global function for getting access to the one and only application object
- `App.GetTopWindow`: This gets the main Top-Level Window in the application

Clicking on the `Button` will cause the `OnButton` method to be called. In `OnButton`, there are examples that show how to use each of the above methods. Each of them will return a reference to a GUI object. In our example, calling `GetChildren` on the `Panel` will return a list of its children controls. Iterating over this list, we print out each of the children, which will just be the Button in this case. `FindWindowById` can be used to find a specific child control; again, we called this on our `Panel` to find the `Button` control. Just to show that we found the `Button`, we used its `SetLabel` method to change its label. Next, calling `GetParent` on the `Button` will return the `Button`'s parent, which is the `Panel`. Finally, by using the global `GetApp` function, we can get a reference to the application object. The `App` object's `GetTopWindow` will return a reference to our Frame.

There's more...

Here are a few more useful methods available for getting references to controls.

Function Name	Description
`wx.FindWindowByLabel(label)`	Finds a child window by looking for it by Label
`wx.FindWindowByName(name)`	Finds a child window by looking for it by Name
`wx.GetTopLevelParent()`	Gets the Top-Level Window, which is at the top of the given control's parental hierarchy

See also

- The *Understanding the window hierarchy* recipe in this chapter outlines the structure of how windows are contained within and are related to each other.

Using Bitmaps

It's likely that, at some point, you will want to be able to display an image in your application. A `Bitmap` is the basic data type that is used to display images in an application. This recipe will show how to load an image file into a `Bitmap` and then display it in a `Frame`.

How to do it...

To see how to use Bitmaps, we will create a little application that loads an image from the hard disk and displays it in a Frame:

```
import os
import wx

class MyApp(wx.App):
    def OnInit(self):
        self.frame = MyFrame(None, title="Bitmaps")
```