wxPython in Action

NOEL RAPPIN, ROBIN DUNN

Перевод: Володеев Сергей

Глава 1. Добро пожаловать в wxPython

Эта глава включает

- Быстрый старт с wxPython
- Создание минимальной wxPython программы
- Импорт модулей wxPython
- Язык программирования Python
- Инструментарий wxWidget
- Соединяем Python и wxWidget

Вот - простая wxPython программа. Она создает окно с одним текстовым полем, которое отображает координаты курсора мыши. Это 20 строк кода, считая пустые строки.

Листинг 1.1 Работающая wxPython программа из 20 строк

```
#!/bin/env python
import wx
class MyFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, -1, "My Frame", size=(300, 300))
        panel = wx.Panel(self, -1)
        panel.Bind(wx.EVT_MOTION, self.OnMove)
        wx.StaticText(panel, -1, "Pos:", pos=(10, 12))
        self.posCtrl = wx.TextCtrl(panel, -1, "", pos=(40, 10))

    def OnMove(self, event):
        pos = event.GetPosition()
        self.posCtrl.SetValue("%s, %s" % (pos.x, pos.y))

if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = MyFrame()
    frame.Show(True)
    app.MainLoop()
```

Что можно сказать о программе из листинга 1.1? Она достаточно короткая и делает не слишком много, тем не менее, она создает окно, заполняет его данными, реагирует на события мыши — это не плохо для программы из 20 строк. Этот пример легко мог быть в три или четыре раза длиннее в некоторых, более кофейных, языках программирования. На рисунке 1.1 изображена программа во время выполнения.

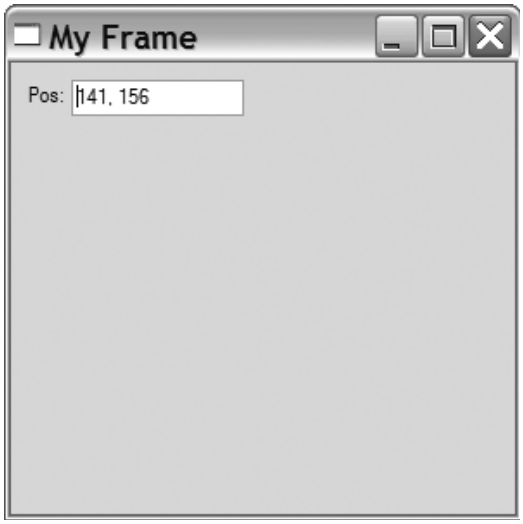

Рисунок 1.1 Наша первая wxPython программа, показывающая координаты мыши

Код читается достаточно легко. Даже если вы не знаете детали языка Python или wxPython, но имеете опыт программирования интерфейсов, вы, вероятно догадаетесь, что означают слова Frame, __ init __, EVT_MOTION, TextCtrl, и MainLoop. Если вы не программируете на Python, могут возникнуть вопросы (например, где открывающие и закрывающие скобки для блоков кода?), и вероятно вы не знаете, что означают некоторые параметры (что это за константы -1?), тем не менее можно понять большую часть кода без подробных объяснений.

В этой книге, мы покажем, почему использование wxPython - это один из самых легких и мощных способов строить реальные программы с GUI. Одни наборы инструментов позволяют легко строить интерфейс непосредственно (в стиле Visual Basic), но не имеют языка программирования с ясностью, гибкостью, и мощью Python. Другие наборы инструментов имеют функциональные возможности wxPython, но вынуждает вас использовать язык неподходящий для быстрой разработки. В wxPython вы найдете правильное сочетание простоты и гибкости максимально эффективное для быстрой разработки программ. К тому же, wxPython – это проект с открытым исходным кодом.

К тому времени, когда вы достигните конца этой книги, вы будете знать, как строить современный GUI с использованием wxPython. Вы будете в состоянии работать с базовыми элементами интерфейса, такими как кнопки и меню, и более продвинутыми элементами интерфейса, типа деревьев и редакторов HTML. В этой главе, мы начнем работать с wxPython, и обсудим, почему инструментарий wxPython будет удачным выбором для ваших программных проектов.

Хороший интерфейс пользователя позволяет обращаться к функциональным возможностям приложения настолько просто насколько это возможно, и имеет элегантный, привлекательный внешний вид. Плохой интерфейс может препятствовать пользователям находить функциональные возможности в программе, и даже может заставить людей предполагать, что совершенно рабочая программа работает с ошибками. В wxPython вы можете создать такой интерфейс, какой захотите и с меньшими усилиями, чем ожидаете.

1.1 Быстрый старт с wxPython

Мы начнем с разработки реальной, хотя и простой, wxPython программы. Пока мы не будем создавать ничто сложного. Мы собираемся шаг за шагом провести вас через весь процесс создания вашей самой первой wxPython программы. Вначале удостоверьтесь, что установили все необходимое. В таблице 1.1 перечислено все, что нужно для выполнения wxPython.

Таблица 1.1 Что должно быть установлено на вашем компьютере для работы wxPython

Инструмент	**Описание**
Правильная операционная система	Это – просто. У вас бгатый выбор: • Любая 32-битная операционная система Microsoft Windows начиная с Windows 98 и выше. • Любая Unix или Linux система, с установленным Gnome Toolkit (GTK). • Macintosh, с операционной системой Mac OS X 10.2.3 или выше.
Язык программирования Python	Доступен для загрузки с www.python.org. Любая версия 2.3 или выше будет работать. Большинство дистрибутивов Linux включают версию Python, как и Mac OS X 10.2.3 или выше. Тем не менее, желательно загрузить последнюю версию.
Инструментарий wxPython	Доступен для загрузки с www.wxpython.org. Есть различные версии, в зависимости от вашей операционной системы и версии Python. Убедитесь, что вы загрузили инсталлятор, который соответствует вашей платформе, версии Python, и предпочтению Unicode. Также загрузите демонстрационный пакет и документацию. Если Вы уже устанавливали какое-либо программное обеспечение на вашу систему, то установка wxPython выполняется аналогично. Последние версии Mac OS X и Linux уже включают wxPython, но лучше загрузить последнюю версию с сайта.
Редактор текста	Мы рекомендуем редактор, который понимает синтаксис Python и может выделять цветом ключевые слова, чтобы сделать код более читаемым. Большинство популярных редакторов включают поддержку Python, так что используйте редактор, который Вы предпочитаете. Если у Вас нет особых предпочтений, то попробуйте IDLE. Это интегрированная среда разработки, которая идет вместе с Python. Она включает редактор исходного текста, интерактивную оболочку, отладчик, и другие инструменты. На сайте Python представлен целый список специализированных Python-редакторов www.python.org/editors.

Все нужное установлено, можно начинать. Мы собираемся создать программу, которая отображает графический файл. Процесс состоит из трех шагов:

1. Мы начнем с пустой минимальной программы wxPython, требуемой для работы.
2. Далее мы сделаем код более структурированным и сложным.
3. И закончим мы версией, которая будет отображать эмблему wxPython.

Рисунки 1.2, 1.3, и 1.4 показывают, на что будет похожа ваша программа в зависимости от используемой платформы.

Рисунок 1.2, hello.py на Windows

Рисунок 1.3, hello.py на Linux

Рисунок 1.4, hello.py на Mac OS X

1.2 Создание пустой минимальной программы wxPython

Давайте начнем с самой простой исполняемой программы на wxPython. Создайте файл, названный “bare.py” и наберите следующий код. Помните, в Python, отступ в начале строки имеет значение.

```
import wx

class App(wx.App):

    def OnInit(self):
        frame = wx.Frame(parent=None, title='Bare')
        frame.Show()
        return True

app = App()
app.MainLoop()
```

Данная программа просто отображает пустое окно. Это не много, но потерпите, скоро мы ее усовершенствуем, и программа будет делать что-нибудь более полезное.

Главная цель этой программы состоит в том, чтобы удостовериться, что вы можете создать исходный файл Python и проверить, что wxPython установлен должным образом. Так что: создайте файл, наберите код, сохраните файл с названием "bare.py", запустите его на выполнение, и убедитесь, что у вас все работает.

Механизм запуска программы зависит от вашей операционной системы. Вы можете выполнить эту программу, набрав ее как параметр командной строки интерпретатора Python, используя одну из следующих команд:

```
python bare.py
```

```
pythonw bare.py
```

На рисунках 1.5, 1.6, и 1.7 показано, как выглядит программа во время выполнения на различных операционных системах.

Рисунок 1.5, bare.py на Windows.

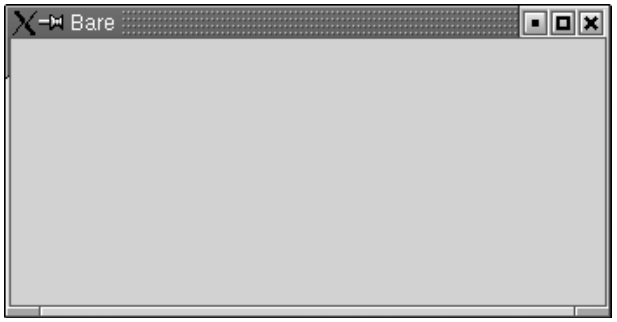

Рисунок 1.6, bare.py на Linux.

Рисунок 1.7, bare.py на Mac OS X.

Окно (window), фрейм (frame), виджет (widget)

Большинство людей глядя на эту программу во время выполнения, сказали бы, что они видят «окно» (“window”). Однако, в wxPython это называется «фрейм» (“frame”). В wxPython, “window” – это базовый класс для любого объекта, который отображается на экране (в некоторых инструментариях это называется виджет (“widget”)). Так, программист wxPython будет часто обращаться к объектам, типа кнопок или текстовых полей как “ windows.” Это кажется немного запутанным. В этой книге, чтобы избежать путаницы, мы будем использовать термин виджет (widget) для базового класса элементов интерфейса. К тому же windows - это название главного продукта большой корпорации. Когда мы будем обращаться к операционной системе подобного названия, мы сделаем это с прописной буквой “W”.

Пока эта программа только создает и отображает пустой фрейм, весь код является основным; удалите любую стоку кода, и программа не будет работать. Эта минимальная

wxPython программа иллюстрирует пять основных шагов, которые нужно сделать при разработке любой wxPython программы:

1. Импорт необходимых пакетов wxPython
2. Создание класса производного от App (application)
3. Определение метода инициализации
4. Создание прикладного объекта
5. Запуск главного цикла приложения

Исследуем эту минимальную программу, и посмотрим, что происходит на каждом шаге.

1.2.1 Импорт пакетов wxPython

Первое, что нужно сделать - импортировать главный пакет wxPython, который называется wx:

```
import wx
```

Как только этот пакет импортирован, вы можете обращаться к классам, функциям, и константам wxPython, используя название пакета wx как приставку, например:

```
class App(wx.App):
```

Импорт в старом стиле

Во время написания этой книги название пакета wxPython изменилось. Так как старое соглашение обозначения все еще поддерживается, вы вероятно столкнетесь с кодом wxPython, написанным в старом стиле. Мы сделаем краткое отступление, чтобы объяснить старый стиль и почему он был изменен. Старое название пакета было wxPython, и он содержал внутренний модуль wx. Было два общих способа импортировать необходимый код — вы могли импортировать wx модуль из пакета wxPython:

```
from wxPython import wx   # УСТАРЕВШИЙ СТИЛЬ!!! НЕ ИСПОЛЬЗОВАТЬ!!!
```

Или, вы могли импортировать все из wx модуля непосредственно.

```
from wxPython.wx import *   # УСТАРЕВШИЙ СТИЛЬ!!! НЕ ИСПОЛЬЗОВАТЬ!!!
```

Оба метода импорта имели серьезные недостатки. Использовать второй метод import * вообще не рекомендуется в Python из-за возможности конфликтов пространств имен. Старый wx модуль избегал этой проблемы, добавляя приставку wx почти ко всем именам. Но это не было надежной гарантией, import * все еще мог вызвать проблемы. Многие программисты wxPython предпочли именно этот стиль, и вы увидите, что он использовался в старом коде довольно часто. Неприятной особенностью этого стиля было то, что названия класса начинались с буквы в нижнем регистре, в то время как большинство wxPython методов начинается с буквы в верхнем регистре, что является полной противоположностью соглашениям принятым для программирования на Python.

Однако, если бы вы захотели избежать конфликтов имен, вызваннх import *, используя from wxPython import wx, то теперь вы должны были бы напечатать “wx” дважды для каждого класса, функции, или константы один раз как приставка пакета и один раз как "нормальная" приставка, например, wx.wxWindow. Это не прижилось. Многие wxPython программисты видели эту дилемму как бородавку, которая должна быть удалена, и в конечном счете, это было сделано.

Еще одна вещь, которую вы должны знать об импортировании wxPython: вы должны импортировать модуль wx прежде, чем вы импортируете что-нибудь еще из wxPython. Вообще, порядок импорта в Python является несущественным, вы можете импортировать

модули в любом порядке. Однако, wxPython, хотя он и похож на единственный модуль, является фактически сложным набором модулей (многие из которых автоматически сгенерированы SWIG), которые являются обертками вокруг функциональных возможностей, обеспеченных основным инструментарием wxWidgets C++ (мы обсудим wxWidgets более подробно в разделе 1.7). Когда вы импортируете модуль wx впервые, wxPython выполняет некоторую инициализацию, которая является жизненно важной для других модулей wxPython. В результате некоторые из модулей wxPython, типа xrc, не могли бы работать должным образом, если wx модуль не был уже импортирован:

```
import wx # Always import wx before
from wx import xrc # any other wxPython packages,
from wx import html # just to be on the safe side.
```

Это требование применяется только к модулям wxPython; другие модули Python вы можете импортировать в любом порядке до или после модулей wxPython. Например, следующий код является правильным:

```
import sys
import wx
import os
from wx import xrc
import urllib
```

1.2.2 Работа с объектами приложение и фрейм

Как только вы импортировали модуль wx, вы можете создать прикладной объект и фрейм. Каждая wxPython программа должна иметь один прикладной объект и по крайней мере один фрейм. Эти объекты подробно обсуждаются в главе 2. Пока, вы только должны знать, что прикладной объект должен быть создан на базе класса wx.App или класса производного от wx.App, где вы должны определить метод OnInit(). Метод OnInit() вызывается wx.App при запуске приложения.

Производный класс от wx.App

Вот так мы определили наш производный класс от wx.App:

```
class MyApp(wx.App):

    def OnInit(self):
        frame = wx.Frame(parent=None, id=-1, title="Bare")
        frame.Show()
        return True
```

Мы назвали наш класс "MyApp", что является общим соглашением, но подошло бы любое правильное название класса Python.

OnInit() - метод, где вы чаще всего будете создавать фреймы (frame). Но обычно вы не будете непосредственно создавать объекты класса wx.Frame, как мы сделали здесь. Вместо этого вы определите ваш собственный класс производный от wx.Frame, также как мы определили производный класс от wx.App. (Мы покажем пример в следующем разделе.) Подробно мы исследуем фреймы в следующей главе, а пока мы просто заметим, что конструктор wx.Frame принимает несколько параметров. Из них обязательным является только первый, остальные имеют значения по умолчанию.

Вызов метода Show() делает фрейм видимым. Если бы мы пропустили его, то фрейм создался бы, но мы бы его не увидели. Мы можем переключать видимость фрейма, вызывая метод Show() с логическим параметром:

```
frame.Show(False) # Make the frame invisible.
frame.Show(True) # True is the default parameter value.
```

```
frame.Hide() # Equivalent to frame.Show(False).
```

Определение конструктора для прикладного объекта

Заметьте, что мы не определяли конструктор __ init __ () для нашего прикладного класса. В Python, это означает что при создании объекта автоматически будет вызван конструктор базового класса wx.App. __init__(). Если вы определяете собственный конструктор __init__(), не забывайте вызвать __init__() базового класса, как в этом примере:

```
class App(wx.App):

    def __init__(self):
        # Call the base class constructor.
        wx.App.__init__(self)
        # Do something here...
```

Если вы забудете так сделать, то wxPython не будет инициализирован и ваш метод OnInit() не будет вызван.

Создание прикладного объекта и запуск главного цикла обработки событий

На заключительном шаге нужно создать объект класса wx.App или производного от него, и вызвать его метод MainLoop():

```
app = App()
app.MainLoop()
```

Все. Как только запускается главный цикл обработки событий, управление переходит к wxPython. В отличие от процедурных программ, wxPython программа GUI управляемая событиями. Главным образом программа реагирует на действия пользователя и обрабатывает события мыши и клавиатуры. Когда все фреймы в приложении будут закрыты, произойдет выход из метода MainLoop(), и программа завершится.

1.3 Дополнения к минимальной программе wxPython

Мы показали вам минимальную wxPython программу, чтобы дать вам удобный старт, что-то маленькое и ничего не делающее, но полезное для обсуждения. Упрощая код, мы написали программу, которую просто понять, но трудно расширять. Мы не рекомендовали бы вам создавать серьезные wxPython программы на ее основе.

Теперь мы собираемся расширить эту минимальную программу. Она пока не богата по функциональным возможностям, но составлена с учетом стандартов программирования Python, и может служить хорошей основой для ваших собственных программ. На листинге 1.2 показана следующая версия, которую мы назвали spare.py.

Листинг 1.2 Дополненная версия нашей минимальной программы.

```
#!/usr/bin/env python    (1)

"""Spare.py is a starting point for a wxPython program."""   (2)

import wx

class Frame(wx.Frame):   (3)
    pass

class App(wx.App):

    def OnInit(self):
        self.frame = Frame(parent=None, title='Spare')   (4)
        self.frame.Show()
        self.SetTopWindow(self.frame)   (5)
```

```
        return True

if __name__ == '__main__':  (6)
    app = App()
    app.MainLoop()
```

Эта версия все еще является маленькой, только 14 строк кода, но мы добавили несколько важных элементов, которые приближают нас к тому, что мы назвали бы хорошим, надежным кодом.

(1) Первая строка в файле похожа на комментарий Python. На некоторых операционных системах, типа Linux и Unix, эта строка говорит операционной системе, где находится интерпретатор, который выполнит файл программы. Если этому файлу программы дать привилегии на выполнение (например, используя команду chmod), то мы могли бы выполнить программу из командной строки операционной системы, просто набрав имя программы:

```
% spare.py
```

Эта строка является удобным соглашением для пользователей Unix и Mac OS X, а на других платформах она просто игнорируется. Даже если вы не используете одну из этих систем, желательно включить эту строку в скрипт, который мог бы быть выполнен на другой платформе.

(2) Мы добавили в модуль строку документации. Если первый оператор в модуле - строка, то эта строка становится строкой документации для модуля и сохраняется в атрибуте модуля __doc__. Вы можете обратиться к строке документации из вашего кода, из некоторых сред разработки, и даже из интерпретатора Python, выполняющегося в интерактивном режиме:

```
>>> import spare
>>> print spare.__doc__
Spare.py is a starting point for simple wxPython programs.
>>>
```

Строки документации - всего лишь один пример мощных способностей самоанализа Python, и мы советуем вам использовать их для модулей, классов, методов и функций. Среды разработки для Python, например PyCrust, способны использовать строки документации, для отображения подсказки во время кодирования.

(3) Мы изменили способ, которым мы создали объект фрейм. В версии "bare" просто создавался объект класса wx.Frame. В версии "spare" мы определили наш собственный класс Frame производный от wx.Frame. С точки зрения конечного результата ничего не изменилось. Но вам будет нужен ваш собственный класс Frame, если вы хотите чтобы на фрейме появилось что-нибудь интересное, типа текста, кнопок и меню. Определение вашего собственного класса Frame готовит почву для будущих изменений. Фактически, как только ваш класс Frame станет достаточно сложным, вы вероятно захотите переместить его в собственный модуль и импортировать этот модуль в вашу главную программу.

(4) Мы добавили ссылку на объект фрейм как атрибут объекта приложения. Снова, мы готовим почву для будущих изменений, а так же демонстрируем, как просто добавить признаки к классам Python. Не имеет значения, что признак - ссылка на сложный, графический объект, типа фрейма. В Python все является объектом.

(5) В методе OnInit() мы вызываем SetTopWindow (метод класса App), передавая в качестве параметра наш недавно созданный объект фрейм. Нам не нужно определять метод SetTopWindow(), потому что он был унаследован от базового класса wx.App. Этот

метод позволяет указать wxPython, какой фрейм или диалог считать главным. Программа wxPython может иметь несколько фреймов, но один из них должен быть определен как главное окно приложения. В нашем случае выбор прост, так как мы имеем только один фрейм.

(6) Заключительное дополнение представляет общий прием, используемый в программах Python. Чтобы проверить, выполняется ли модуль как программа или был импортирован другим модулем, нужно проверить атрибут модуля __name__:

```
if __name__ == '__main__':
    app = App()
    app.MainLoop()
```

Если модуль был импортирован, то его атрибут __name__ будет совпадать с именем файла (без расширения):

```
>>> import spare
>>> spare.__name__
'spare'
>>>
```

Но если модуль выполняется, а не импортируется, Python отменяет значение по умолчанию, и присваивает атрибуту модуля __name__ значение '__main__'. Мы используем эту особенность, создавая прикладной объект и запуская главный цикл, только если модуль выполняется как программа.

Если бы мы не выполняли эту проверку, и создавали прикладной объект, даже когда модуль был импортирован, то это могло бы создать проблемы в импортирующем модуле, особенно, если импортируемый модуль уже запустил цикл обработки событий. Причем это было бы весьма трудно проверить (тем более, что в программе wxPython может быть только один прикладной объект, и как только мы запускаем цикл обработки событий, управление переходит к wxPython). Не запуская наше собственное приложение, когда модуль импортирован, мы делаем наш фрейм и классы доступными для других программ Python, облегчая повторное использование существующего кода.

1.4 Создание финальной программы hello.py

Теперь, когда у вас есть основа для финального рывка, давайте создадим заключительную версию программы, которую мы показывали в начале этой главы. Создайте файл hello.py и введите код, который показан в листинге 1.3.

Листинг 1.3 Финальная hello.py

```
#!/usr/bin/env python  (1)

"""Hello, wxPython! program."""

import wx

class Frame(wx.Frame):  (2)
    """Frame class that displays an image."""

    def __init__(self, image, parent=None, id=-1,  (3)
                 pos=wx.DefaultPosition,
                 title='Hello, wxPython!'):
        """Create a Frame instance and display image."""
        temp = image.ConvertToBitmap()  (4)
        size = temp.GetWidth(), temp.GetHeight()
        wx.Frame.__init__(self, parent, id, title, pos, size)
        self.bmp = wx.StaticBitmap(parent=self, bitmap=temp)
```

```
class App(wx.App):  (5)
    """Application class."""

    def OnInit(self):
        image = wx.Image('wxPython.jpg', wx.BITMAP_TYPE_JPEG)  (6)
        self.frame = Frame(image)
        self.frame.Show()
        self.SetTopWindow(self.frame)
        return True

def main():  (7)
    app = App()
    app.MainLoop()

if __name__ == '__main__':  (8)
    main()
```

(1) Первая строка позволяет этой программе быть исполняемым скриптом под Linux и другими Unix-подобными операционными системами.

(2) Определение собственного класса Frame, производного от wx.Frame, позволяет нам легко управлять содержанием фрейма.

(3) Мы добавили параметр image для конструктора нашего класса Frame. Его значение определяется в классе приложения, при создании объекта Frame. Мы должны передать необходимые параметры конструктору wx.Frame. __ init __ (), но нет причин, почему мы не могли бы добавить больше параметров для конструктора нашего класса.

(4) Мы собираемся отображать картинку используя элемент wx.StaticBitmap, который требует bitmap. Поэтому мы преобразуем изображение в bitmap. Мы также создаем кортеж размера, используя ширину и высоту bitmap. Кортеж размера передается в конструктор фрейма wx.Frame.__init__(), так, чтобы размер фрейма соответствовал размеру bitmap.

(5) Определение метода OnInit() - необходимое требование для любого приложения wxPython.

(6) Мы создаем объект-изображение (image), используя файл wxPython.jpg, находящийся в том же каталоге что и hello.py. Вы можете взять этот файл с сайта издательства, или заменить его своим собственным. Более сложная версия этой программы принимала бы название файла картинки из командной строки. Мы передаем наш объект-изображение как параметр при создании фрейма.

(7) Функция main() создает объект-приложение и запускает цикл обработки событий wxPython.

(8) Проверка, является ли этот модуль главным, позволяет использовать его двумя различными способами: выполнить из командной строки или импортировать другим модулем.

Что произошло, когда вы запустили вашу версию hello.py? Вы увидели фрейм, по размеру соответствующий вашему изображению? В противном случае встряхнитесь и попробуйте еще раз. Если все получилось, поздравляем! Вы готовы сделать следующие захватывающие шаги.

Но прежде, чем перейти к следующей главе, мы собираемся поговорить о wxPython более подробно. На что он способным, и благодаря чему это стало возможно. Если это не интересует вас, не стесняйтесь, переходите к следующей главе, а для остальных мы продолжим.

1.5 На что способен wxPython?

Почти весь интерфейс, в котором вы нуждаетесь, может быть создан в wxPython. В этом разделе, мы покажем, на что это похоже, используя изображения элементов интерфейса из демонстрационной версии wxPython. Рисунок 1.8 - сложное изображение, показывающее все основные виджеты, которые вы ожидаете: кнопки, checkboxes, поле со списком, меню, список, spinner control, текстовые поля и радио-кнопки.

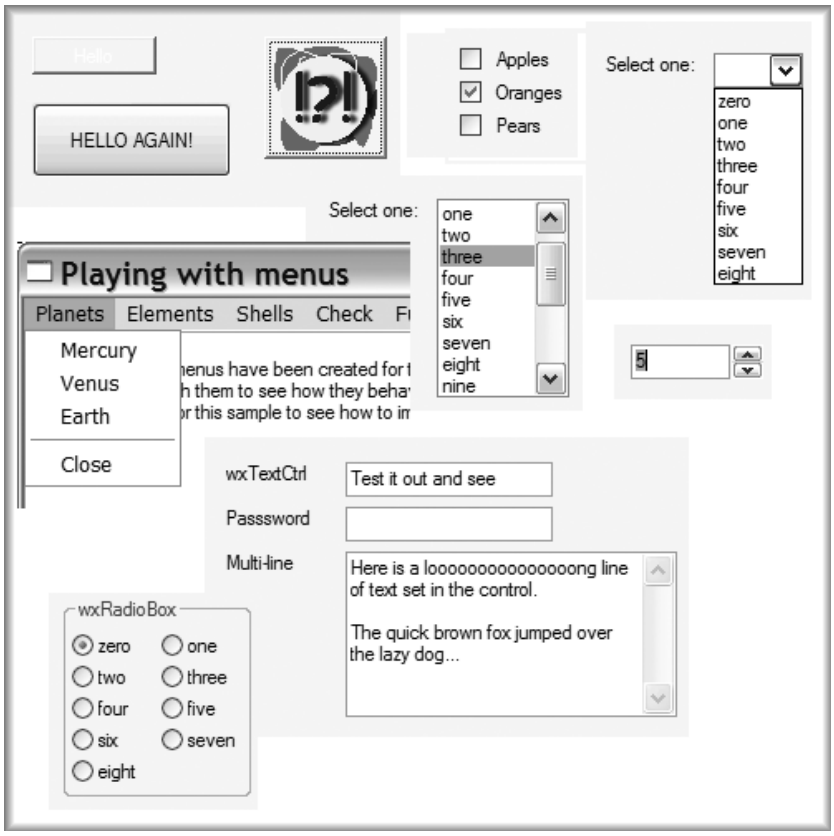

Рисунок 1.8 Базовые элементы интерфейса

На рисунке 1.9 показаны менее распространенные, но тоже очень полезные виджеты, включая slider control, доступный для редактирования список, поле для ввода времени, панель инструментов, закладки, tree list control и аналоговые часы.

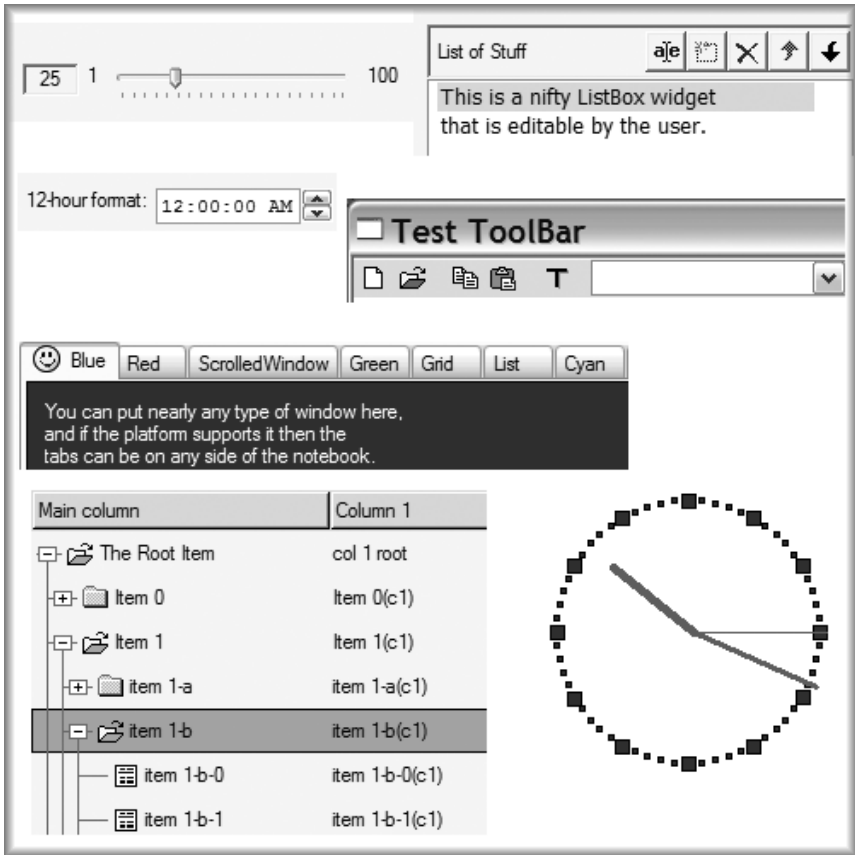

Рисунок 1.9 Дополнительные элементы интерфейса

Grid control (сетка) - один из самых гибких виджетов wxPython, он позволяет настраивать представление и редактирование ячеек. На рисунке 1.10 показаны некоторые возможности сетки.

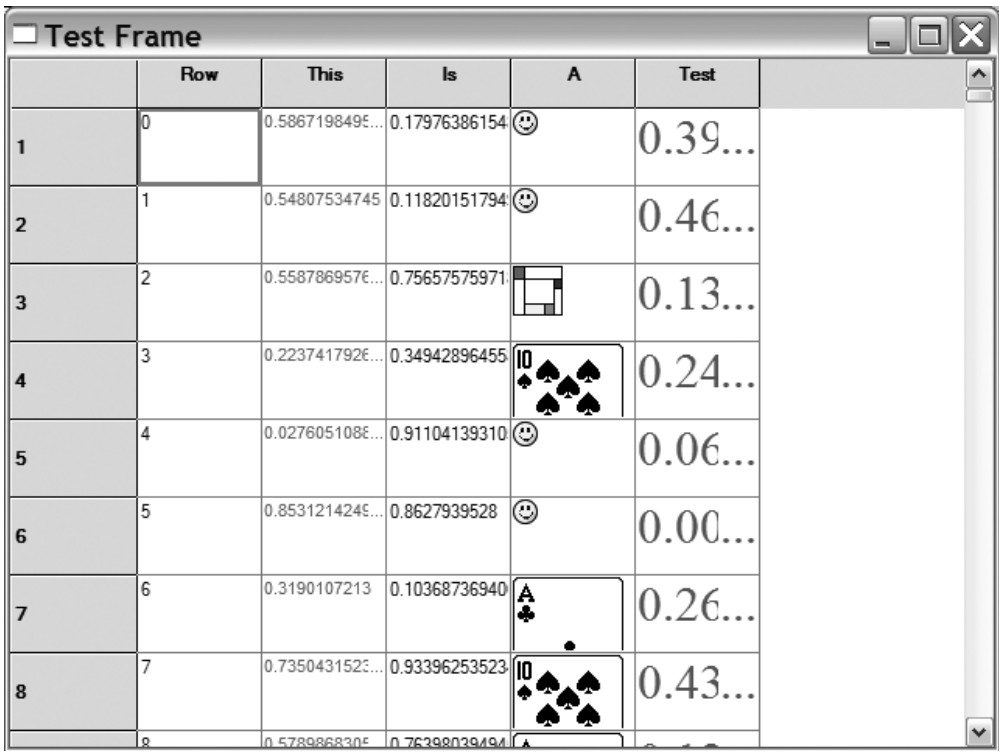

Рисунок 1.10 Пример использования сетки (grid)

И это еще не все! Также вы получаете HTML виджет, который Вы можете использовать для отображения статического форматированного текста, как основу простого web-браузера, как основу справочной системы, или где-то еще, где требуется отображение HTML. Пример использования на рисунке 1.11.

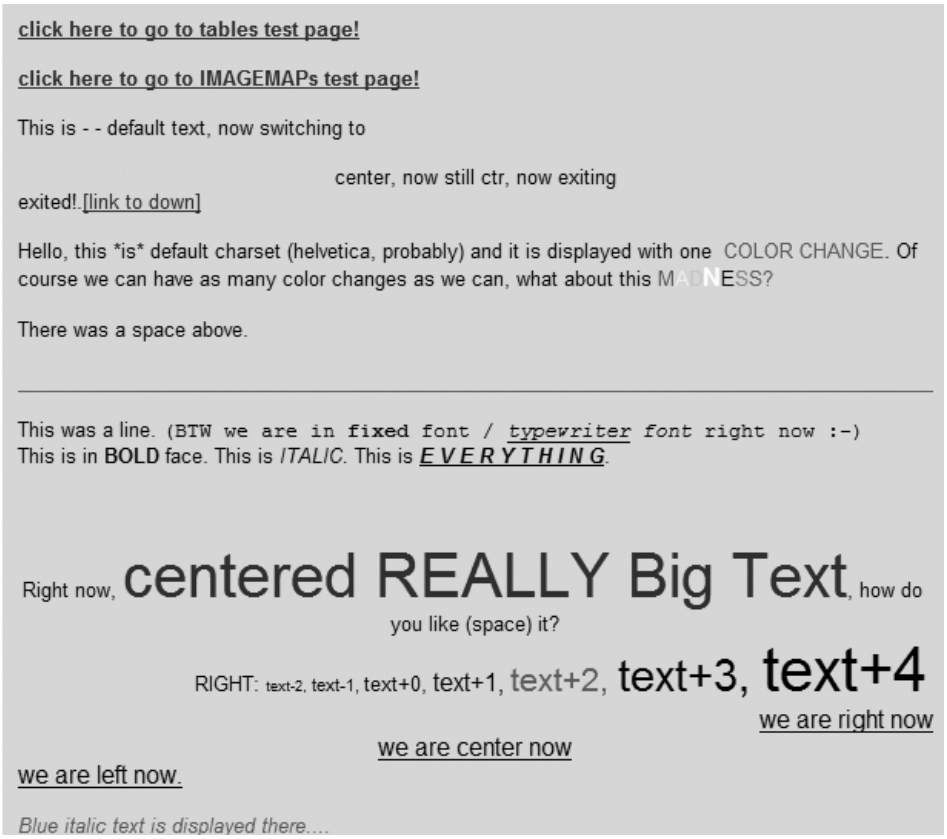

Рисунок 1.11 wx.HTMLWindow, отображает HTML.

И это только поверхностный взгляд. Библиотека wxPython также включает инструменты для анимации изображения. Вы также получаете поддержку буфера обмена и технологии drag-and-drop, поддержку MIME и аудио, все стандартные диалоги, предлагаемые вашей операционной системой, способность хранить определение интерфейса в файле XML, полное управление размещением ваших виджетов, и многое другое.

1.6 Почему выбирают wxPython?

Наибольшую выгоду от wxPython можно получить всесторонне изучив его. В то же время мы думаем, что программистам, которые разрабатывают пользовательский интерфейс (UI), от случая к случаю, также принесло бы пользу использование wxPython.

1.6.1 Программисты Python

Если вы - уже программист Python, вы вероятно заметили, что Tkinter, GUI инструментарий, распространяемый вместе с Python, имеет некоторые проблемы:

- Tkinter основан на инструментарии Tk, который поддерживает несколько устаревший вид виджетов. По умолчанию, он не поддерживает более сложные виджеты, типа деревьев или окон с закладками. Он также не имеет богатого набора предопределенных диалогов.
- Набор инструментов Tk не использует системную поддержку виджетов, приводя к приложению, которое выглядит инородным на всех платформах. В wxPython, диалоги и виджеты не будут отличаться от стандартных для данной операционной системы. Пользователь Tk обнаружит, что кнопки, шрифты, меню и весь вид, немного отличается от того, что он ожидает.
- Многие программисты считают Tkinter несколько неуклюжим. В частности процесс, связи событий с действиями в wxPython, более гибок и мощен.

Вы обнаружите, что в wxPython эти проблемы решены. Инструментарий wxPython значительно более полон и обширен, чем Tkinter, и системная поддержка виджетов

означает, что ваше приложение будет выглядеть как родное в вашей операционной системе. Дополнительно, в wxPython реализована поддержка особенностей языка Python.

1.6.2 Пользователи wxWidget

Если вы уже используете wxWidgets, то wxPython предлагает вам использование языка Python. С его ясным синтаксисом, динамической типизацией и гибкой объектной моделью, Python может многократно улучшить вашу производительность. Python имеет обширную стандартную библиотеку, которая легко включается в ваше приложение. Программы Python имеют тенденцию быть короче и менее подвержены ошибкам чем программы на C++. Есть также множество дополнений к wxWidgets только для Python.

1.6.3 Новые пользователи

Если вы в настоящее время не используете Python и wxWidgets, вы можете получить реальную выгоду от этого обширного набора инструментов и языка Python. Если вы в настоящее время используете Java/ Swing, то вы вероятно найдете wxPython менее сложным и более легким в использовании, и язык Python, значительно менее детальный чем Java. Если вы в настоящее время программируете для единственной платформы на C++ используя Microsoft Foundation Classes (MFC), то вы оцените перспективы кроссплатформенной разработки на wxPython. Однако, чтобы понимать примеры в этой книге, полезно знать основы языка Python. Если вы только начинаете изучать Python, попробуйте The Quick Python Book, by Daryl Harms and Kenneth McDonald, или загляните на сайт www.diveintopython.org.

В следующем разделе, вы узнаете о составляющих частях wxPython: языке Python и инструментарии wxWidgets. Мы также объясненим как правильно использовать wxPython.

1.7 Как работает wxPython

В предыдущем разделе, мы говорили о том, что может сделать wxPython. В этом разделе, мы подробно рассмотрим как wxPython работает. Внутренне, wxPython - обертка или интерфейс для популярного C++ инструментария для создания GUI, под названием wxWidgets. Проект wxWidgets имеет длинную историю. Именно он источник большинства функциональных возможностей wxPython. wxPython позволяет вам получать все преимущества wxWidgets, программируя на Python, а не на C++.

Набор инструментов wxPython - комбинация двух замечательных программных систем, которые имеют более чем 25 летний путь развития. Кроме того, сам wxPython был результатом существенного объема работы. Чтобы заставить wxPython работать, использается инструмент SWIG, для генерации функций оберток и промежуточного кода, которые позволяют программе Python использовать библиотеку C++ wxWidgets так же, как если бы это была одна из библиотек Python. Хотя SWIG делает много работы, есть все еще некоторая часть ручной работы, чтобы сделать доступ к объектам wxPython таким же, как другим объектам Python. Также было создано несколько дополнительных виджетов, написанных непосредственно в wxPython, которые не доступны в версии для C++. Вы столкнетесь с некоторыми из них при чтении этой книги.

В этом разделе мы дадим краткий обзор языка программирования Python и инструментария wxWidgets C++. Это - комбинация непринужденности использования Python и широкого диапазона функциональных возможностей wxWidget, которые дают wxPython его уникальную мощь.

1.7.1 Язык программирования Python

Python - язык программирования, который легко справляется как с небольшими скриптовыми задачами, для которых обычно используют Perl, так и с полномасштабной разработкой приложений, обычно связанной с C ++ или Java. Используя простй, изящный, краткий, синтаксис и чистую, последовательную, семантическую модель, Python позволяет программистам легко комбинировать простые части, чтобы сделать сложное целое.

Далее в остальной части этой книги, предполагается, что вы имеете хорошее практическое знание Python, и знакомы с фундаментальными понятиями типа объектов и классов. Вы не должны быть экспертом Python, чтобы читать эту книгу, но обычные языковые конструкции Python не объясняются при обсуждении примеров wxPython. Если вы нуждаетесь в большем количестве вводной информации по Python, сайт Python содержит превосходные учебники и другую документацию (www.python.org/doc).

Одна важная особенность. Python – это интерактивный интерпретатор, который может быть очень полезным в исследовании языка и в отладке программ. Если Python установлен в доступном каталоге, вы можете обратиться к интерпретатору, вводя python в командной строке. После этого вы увидите подсказку >>>, которая является приглашением к вводу команд Python. После этого, вы можете ввести любое выражение Python, и его значение будет отображено на экране. Например:

```
$ python
Python 2.3.3c1 (#50, Dec 4 2003, 21:27:34) [MSC v.1200 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
>>> 10 / 3
3
>>> zip(['a', 'b', 'c'], [1, 2, 3])
[('a', 1), ('b', 2), ('c', 3)]
>>>
```

В этом коротком сеансе, я сделал несколько простых арифметических действий, затем использовал встроенную функцию zip(), которая объединяет два списка в связанный список. Вы можете делать в интерпретаторе все, что вы делаете в автономной программе Python: импортировать модули, определять функции, определять классы и т.д.

1.7.2 Инструментарий wxWidgets

Другой основной компонент wxPython – инструментарий wxWidgets. В основе, wxWidgets – библиотека GUI, реализованная на C++. Это набор классов C++, в которых заложены широкие возможности. wxWidgets также содержит некоторые полезные вещи не связанные с пользовательским интерфейсом, например структуры данных, не поддерживаемые стандартом ANSI C++, типа строк и хэштаблиц. Так как эти и другие особенности уже реализованы в языке Python или его стандартной библиотеке, обертки для этих классов wxWidgets не поддерживаются в wxPython, и вы должны использовать эквиваленты Python вместо них. Главным образом wxPython обеспечивает только обертки для классов wxWidgets связанных с GUI. Цель wxWidgets состоит в том, чтобы позволить программам на C++ компилироваться и выполняться на всех поддерживаемых платформах.

Вот пример C++ wxWidgets программы, взятый из учебника Роберта Роеблинга с сайта wxWidgets. В программе создается окно и меню с двумя пунктами: Quit и About. Эта программа приведена для сравнения с примерами на Python, которые вы увидите далее в этой книге.

Листинг 1.4 Простая программа Hello World в C++ wxWidgets

```
#include "wx/wx.h"

class MyApp: public wxApp {
    virtual bool OnInit();
};

class MyFrame: public wxFrame {
public:
    MyFrame(const wxString& title, const wxPoint& pos,
            const wxSize& size);
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
    DECLARE_EVENT_TABLE()
};

enum {
    ID_Quit = 1,
    ID_About,
};

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(ID_Quit, MyFrame::OnQuit)
    EVT_MENU(ID_About, MyFrame::OnAbout)
END_EVENT_TABLE()

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit() {
    MyFrame *frame = new MyFrame("Hello World", wxPoint(50,50),
    wxSize(450,340));
    frame->Show(TRUE);
    SetTopWindow(frame);
    return TRUE;
}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos,
                 const wxSize& size)
: wxFrame((wxFrame *)NULL, -1, title, pos, size) {
    wxMenu *menuFile = new wxMenu;
    menuFile->Append( ID_About, "&About..." );
    menuFile->AppendSeparator();
    menuFile->Append( ID_Quit, "E&xit" );
    wxMenuBar *menuBar = new wxMenuBar;
    menuBar->Append( menuFile, "&File" );
    SetMenuBar( menuBar );
    CreateStatusBar();
    SetStatusText( "Welcome to wxWidgets!" );
}

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event)) {
    Close(TRUE);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event)) {
    wxMessageBox("This is a wxWidgets Hello world sample",
    "About Hello World", wxOK | wxICON_INFORMATION, this);
}
```

Если вы знакомы с C ++, вы вероятно заметили, что кое-что отсутствует. Обычно, C++ программы имеют функцию main (), которая является отправной точкой для программы. В wxWidgets, макрос IMPLEMENT_APP (MyApp) автоматически устанавливает функцию main() и управляет инициализацией wxWidget программы.

Как и в большинстве кроссплатформенных библиотек для создания интерфейса, классы и методы, видимые программисту - фактически являются обертками для ряда производных

классов. Типично есть производный класс для каждой платформы, под которой выполняется wxWidgets, и именно он автоматически используется для текущей платформы. Вот список наиболее существенных поддерживаемых платформ:

- Microsoft Windows
- Mac OS
- Gnome Toolkit (GTK+), which is applicable on most modern Unix systems

Для каждой платформы, wxWidgets пытается использовать родные виджеты и особенности, где это возможно, и вообще пытается подражать родному интерфейсу каждой платформы. Таким образом, wxWidgets избегает проблемы «наименьшего общего знаменателя», которую часто имеют другие инструментарии кроссплатформенной разработки.

Если вы работали с любым другим крупномасштабным объектно-ориентированным инструментарием для создания интерфейса, типа MFC или Java Swing, то основная структура wxWidgets должна показаться вам знакомой. Одно отличие от некоторых наборов инструментов состоит в том, что wxWidgets не делает различия между виджетами, которые могут содержать другие виджеты и теми, которые не могут. (например, в Java Swing это классы JComponent и JContainer). Механизм для добавления дочерних виджетов встроен в базовый класс wxWindow так, чтобы потенциально это было доступно для всех виджетов. В принципе все виджеты можно воспринимать как как контейнеры. Хотя, как правило, виджеты не являющиеся контейнерами по своей сути, препятствуют вам использовать это поведение (например, вы не можете поместить диалоговое окно в кнопку).

Разработка wxWidgets была начата раньше, чем вы могли бы предположить. Проект начал Julian Smart в 1992 году, в University of Edinburgh's Artificial Intelligence Applications Institute. Smart пробовал строить приложение, которое могло бы выполниться и на Unix и на Windows. Существующие коммерческие наборы инструментов были предельно дороги, поэтому он написал свой собственный инструментарий. Название wxWidgets содержит ссылки на две оригинальные платформы — "w" для Microsoft Windows и "x" для X сервера Unix. Оригинальная версия была написана в MFC, для версии Windows, и в XView для Unix. Но их быстро заменили более общие библиотеки для каждой платформы. XView был заменен набором инструментов Motif, и MFC был заменен прямыми вызовами функций Windows API. В 1997, вся система была построена с более гибким API, и GTK+ версия стала стандартным портом для Unix. Портирование на Macintosh было осуществлено в следующем году.

Python не единственный язык, который имеет библиотеку для поддержки wxWidgets, хотя он и имеет самое многочисленное пользовательское сообщество. На сайте wxWidgets есть ссылки на проекты, в которых реализована поддержка таких языков как Ada, Basic, C#, Eiffel, Euphoria, Haskell, Java, JavaScript, Lua, Perl, и Ruby, хотя мы не делаем никаких предположений относиттельно надежности или уровня поддержки любого из этих портов.

1.7.3 Соединение всего этого: wxPython

В то время как и Python и wxWidgets являются довольно большими самостоятельными продуктами, при их объединении создается еще большее целое. Гибкость языка Python делает wxPython более легким для разработки чем C++, в то же время код wxWidgets написанный на C++ дает Python скорость и доступ к системным элементам интерфейса. Таблица 1.2 дает представление о некоторых проблемах, которые являются сложными в C++, и простыми, если не тривиальными, в Python.

Таблица 1.2 Разработка в C++ против разработки в wxPython

C++	**wxPython**

Распределением памяти управляет программист	Распределением памяти управляет Python
Статическое связывание делает трудным полиморфизм	Динамическое связывание облегчает полиморфизм
Program reflection very limited	Program reflection easy, allowing for powerful abstraction
Нельзя легко использовать функции как параметры	Функции может быть передана в качестве параметра как любая другая переменная
Цикл компиляции необходим перед каждым выполнением	Программа интерпретируется во время выполнения

Вот пример того, как взаимодействуют эти два инструмента. В предыдущем разделе, мы показали вам пример “hello world” в C++ wxWidgets. В листинге 1.5 показан тот же самый пример, переведенный на wxPython.

Листинг 1.5 Простая программа Hello World в wxPython

```
import wx

class MyApp(wx.App):

    def OnInit(self):
        frame = MyFrame("Hello World", (50, 60), (450, 340))
        frame.Show()
        self.SetTopWindow(frame)
        return True

class MyFrame(wx.Frame):

    def __init__(self, title, pos, size):
        wx.Frame.__init__(self, None, -1, title, pos, size)
        menuFile = wx.Menu()
        menuFile.Append(1, "&About...")
        menuFile.AppendSeparator()
        menuFile.Append(2, "E&xit")
        menuBar = wx.MenuBar()
        menuBar.Append(menuFile, "&File")
        self.SetMenuBar(menuBar)
        self.CreateStatusBar()
        self.SetStatusText("Welcome to wxPython!")
        self.Bind(wx.EVT_MENU, self.OnAbout, id=1)
        self.Bind(wx.EVT_MENU, self.OnQuit, id=2)

    def OnQuit(self, event):
        self.Close()

    def OnAbout(self, event):
        wx.MessageBox("This is a wxPython Hello world sample",
                "About Hello World", wx.OK | wx.ICON_INFORMATION, self)

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

Есть два момента, на которые мы хотели бы обратить ваше внимание, при сравнении примеров wxPython и wxWidgets C++ (помимо простого различия между этими двумя языками).

Во первых, заметьте, что wxPython не имеет автоматического макроса для создания функции main, и должен сделать это явно в конце модуля.

Во вторых, механизм, которым связываются события с кодом, отличается в этих программах. Так как Python позволяет функциям легко передаваться в качестве

параметров, wxPython программа может использовать относительно простой метод wx.Bind(), который динамически связывает событие и обработчик во время выполнения. C++ программа должена использовать макросы DECLARE_EVENT_TABLE и BEGIN_EVENT_TABLE, которые делают статическое связывание, что является несколько более неуклюжим.

Без учета этих особенностей, эти две программы очень похожи. И все же, мы считаем, что версия Python читатся легче. Преимущества Python проявляются более ярко в больших программах, из-за его более простого синтаксиса, автоматического управления памятью, и т.д. Здесь стоит отметить, что wxPython появился не случайно. Он был разработан, чтобы удовлетворить определенную потребность в кроссплатформенной среде быстрой разработки. И он развивается благодаря длительным усилиям программистов, которые нуждаются в быстрой разработке GUI.

Разработка wxPython и wxWidgets продолжается. Новые проекты включают поддержку мобильных устройств и лучшую поддержку мультимедиа. Актуальная версия wxPython доступна на www.wxpython.org.

1.8 Резюме

- Вы можете создать минимальную wxPython программу меньше чем из 10 строк. Большинство wxPython программ намного длиннее 10 строк, и обычно разбиваются на отдельные модули, содержащие определение классов производных от классов wxPython, включающих, как мы надеемся, большое количество строк документации.

- К большинству функций wxPython обращаются используя пакет wx, который должен быть импортирован оператором import wx. Каждая wxPython программа должна иметь прикладной объект класса производного от wx.App, в котором определяется метод OnInit(). Большинство wxPython программ будет иметь один или более фреймов – объектов классов производных от wx.Frame. Фрейм – объект подобный окну, который появляется на экране, часто с меню, строкой состояния, панелью инструментов и другими виджетами. Управление вашей программой переходит к wxPython, когда ваше приложение вызывает метод MainLoop().

- wxPython предоставляет все основные виджеты, которые вы ожидаете, плюс общие диалоги, широкое разнообразие более сложных виджетов для отображения HTML, сеток в стиле электронных таблиц, и т.д. wxPython основан на wxWidgets - C ++ инструментарии с огромными возможностями. Это кроссплатформенный набор инструментов, поддерживающий Microsoft Windows, Unix GTK + и Mac OS. Основной единицей wxWidgets приложения является окно (window), означая любой элемент, который может быть изображен на экране.

- wxPython – это комбинация языка программирования Python и wxWidgets. Он может быть загружен с сайта www. wxpython.org. Он соединяет обширный набор инструментов интерфейса с удобным в языком создания сценариев. Он предлагает повышение производительности и полезные возможности для любого программиста, включая Python- или wxWidgets-программистов.

- wxPython является оберткой вокруг wxWidgets. Он содержит интерфейсы, которые позволяют языковым конструкциям Python взаимодействовать со структурами C++. Эти обертки в значительной степени созданы благодаря инструменту SWIG, из длинного списка описаний того, как объекты Python и C++ взаимодействуют друг с другом.

Теперь пришло время программировать в wxPython. Следующая глава начинается с написания некоторого кода, и остаток первой части мы исследуем наиболее важные понятия wxPython. Вперед!